FILE COPY

2

AD-A223 155

CECOM

CENTER FOR SOFTWARE ENGINEERING

ADVANCED SOFTWARE TECHNOLOGY

DTIC
ELECTE
JUN 21 1990
S E D

Subject: **Final Report - Real-Time Ada Demonstration Project**

CLEARED
FOR OPEN PUBLICATION
SEP 20 1989
DIRECTORATE FOR FREEDOM OF INFORMATION
AND SECURITY REVIEW (OASD-PA)
DEPARTMENT OF DEFENSE

REVIEW OF THIS MATERIAL DOES NOT IMPLY
DEPARTMENT OF DEFENSE [illegible]
FACTUAL [illegible]

CIN: **C02 092LA 0009 00**

31 MAY 1989

This document has been approved for public release and sale; its distribution is unlimited.

394229

90 06 21 045

DEMONSTRATION PROJECT

FINAL REPORT

DTIC COPY INSPECTED 1

PREPARED FOR:
U.S. Army HQ CECOM
Center for Software Engineering
Advanced Software Technology
Fort Monmouth, NJ 07703-5000

PREPARED BY:
LabTek Corporation
8 Lunar Drive
Woodbridge, CT 06525

DATE:
27 April 1989

Accession For	
NTIS GRA&I	✓
DTIC TAB	☐
Unannounced	☐
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	

(A)

EXECUTIVE SUMMARY

The Ada programming language has been available to software developers for several years, yet its acceptance into the real-time embedded applications for which it was developed ~~[1]~~, has been less than universal. Although many staff software engineers were quick to embrace the language ~~[2]~~, they soon realized that the ability to assert control over the hardware was greatly restricted with the available Ada compiler implementations. The result was a disappointment, and often project delays were due specifically to the use of Ada on the project. Performance of initial compilers (and even many contemporary compilers) was far below that achievable from alternative languages. In the "heat of battle" associated with hardware/software integration, sufficient time was not available to work out the problems with the compilers, and often cumbersome work-arounds were implemented. The impact of these initial costly experiences has served to retard the adoption of Ada for real-time embedded applications, although its use in other applications has exceeded most expectations.

The purpose of this project was to address the difficulties in real-time Ada programming from an "Ada technology" perspective, and to provide accurate details on some of the perceived problems with Ada. The project involves the development of a typical weapon system application with severe performance requirements. The application is synthetic, but resembles many similar weapon systems. In some cases, simplifications were adopted because they did not alter the nature of the application significantly, and may have restricted the ability to make the results of the project available.

(KR)

The approach used was to develop the software to be independent of the target architecture, and to increase the number of processors as necessary to achieve the required system performance. The unit of distribution supported was the Ada task. The project utilized a commercially available Ada compiler and supported the Ada semantics by extending the runtime system without modifying the vendor supplied runtime code. Initially, it was intended to use a "source code translation" method to achieve the distribution. This would translate the input source according to a configuration table and generate individual programs for loading on the target processors. This technique was not used because it was found to be somewhat easier to apply a link/edit step on the output of the compiler to achieve the desired effect. In general, it was felt that any production project should use a compiler that specifically supports the distribution of Ada programs. Full Ada semantics including the Ada rendezvous were preserved across the distributed system so that no source code changes were necessary to alter the allocation of tasks to processors. This allocation was done using a distribution table that specified the object names, the processor ID, and relevant characteristics.

The project has identified areas in the Ada language definition that are vague and need special clarification with respect to distributed systems. Although some of these have been identified previously, no resolutions have been adopted and it is hoped that this work will shed some insights into how they may be resolved in future interpretations/revisions of the language. Furthermore, the results of the demonstration should help to indicate what is achievable using Ada, and how performance can be improved by judicious use of language features. Finally, the project demonstrates that it can be practical to use distributed systems effectively within the Ada model of concurrency and that the difficulty of adding additional processors can be minimized. The work performed on this project will be available to other researchers and compiler writers to aid in understanding and resolving the real-time Ada issues.

The project used the Ada tasking model exclusively to achieve concurrent execution. It was our objective to fully utilize the real-time features of Ada to meet performance objectives. Unfortunately, we found that while the latest compilers are starting to provide these features, they are still quite far from being error free. We spent close to ninety percent of our integration time analyzing problems that were the result of runtime and code generator anomalies. If this had been other than a demonstration project, we certainly would have been forced to alter the design to limit the features used. Application developers should be prepared to spend additional time during integration to resolve such problems if they expect to use Ada as it was intended to be used for real-time systems.

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

List of Figures

Demonstration Project Final Report

1. Introduction

This paper is divided into two separate, but related topics. The first topic covers the details of the application chosen for the demonstration. Since the application was developed with the intention of running on a single CPU, it is essentially divorced from the underlying implementation hardware architecture. The second topic is on the distribution of Ada programs for loosely coupled multiprocessors. This aspect of the project has been an area of considerable debate which is unlikely to subside until several implementations resolve the issues. Unlike tightly coupled processors, loosely coupled processors incur significant overhead in inter-processor communication. For this reason, it has been questionable whether or not the tasking semantics of Ada can be (or should be) implemented for use on real-time systems. This section discusses many of the critical issues in using the Ada model of concurrency across a network and provides the implementation strategy used on the demonstration project.

2. Demonstration Application

The demonstration application was chosen among three proposed candidates.

1) Attitude/Altitude Control for remote controlled helicopter (RPV)
2) Vision-based sentry system to perform guard duty
3) A weapon system to intercept attacking armor

The first candidate was rejected because the risk associated with the mechanics of the system affecting the success of the project was deemed to be unacceptable. The third candidate was selected over the second candidate because it seemed to have greater similarity to a majority of DoD applications, and because of more severe real-time requirements. The selected system was titled the "Border Defense System (BDS)" and is designed to provide short to medium range protection against a massive armored attack.

The complete requirements for the BDS are contained in Appendix A, however the main characteristics are summarized below:

- Hard Deadline Driven application: failure to meet timing requirements will result in total mission failure.

- "Processor in the Loop" flight control with dynamic target tracking.

- Complex problem, with interaction among several different functional areas:

    Message Reception (from Sensor Interface and Airborne Rockets)

    Target Tracking and Prediction (up to 100 simultaneous targets)

    Rocket Tracking and Guidance (up to 20 simultaneous rockets, rockets travel at supersonic velocities and are guided by updates every 100ms)

    Real-Time Graphics Updates

    Real-Time Operator Interface (Mouse peak data rate of 500Hz)

    Message Transmission (to Rockets and Rocket Launcher)

- Using current technology: 32-bit Microprocessors (80386-16MHz)

- A separate program was designated for the simulator initially, however the simulation portion of the project was incorporated into the system as additional tasks and placed on a separate processor using the distribution technique.

- Less than 1.0% "code" statements. No assembly language allowed.

- All application concurrency is expressed using the Ada Tasking Model (Rendezvous) exclusively.

- The program consists of approximately 3200 Ada LOC, of which 26 are Inlined code statements. It is divided into 31 library units or subunits, and contains 12 tasks (1 interrupt entry).

2.1 Design Method

The selected design approach uses an iterative technique to refine the system and software design. It is Time/Risk Driven to address the areas that are perceived most difficult first.

A key component of the method is to prototype those algorithms that are known to be required in the system, yet their execution time is difficult to accurately estimate. The resulting prototype data is used to develop timing budgets and design a software structure to insure correct timing. Emphasis is placed on meeting (software) deadlines first, and to get the exact functionality later. (However functionality must be sufficiently correct to model the timing accurately.) This approach is driven by the (unpleasant) experience that it is often much easier to "fix" the functionality than the performance. Put another way, it can be extremely difficult to improve the performance of a system that is grossly out of specification. Systems that are initially 5 to 20 times too slow are not uncommon, and frequently result in complete redesign. This places timing on top of the RISK list. Where the functionality of some processing is not well understood, these are also appropriate areas to prototype. Prototypes may be utilized in the final product providing that they are upgraded to insure compliance with coding styles. The process associated with the design method used is described as the Ada Time Oriented Method (ATOM).

It should be noted that one possible drawback of any iterative technique is the confusion that can occur in the minds of the designers/implementers with the different variations on the design. Care must be taken not to "brainstorm" at the implementation level for fear of mass confusion resulting. The degree to which a clear state of the system design can be conveyed (either graphically or textually) dictates the freedom with which designs may evolve. To ignore the nature of humans to confuse very similar designs by getting details transposed in their minds, is a prescription for long integration cycles.

2.2 Ada Time Oriented Method (ATOM)

This software design technique is targeted for hard real-time embedded applications, where the most difficult aspect of the project is meeting the timing requirements. In these applications, the correct functioning of the system depends on proper timing as much as the correctness of the calculations. This method may not be appropriate for applications that do not have a significant real-time component.

For the following steps in the method, the term "message" simply refers to an I/O transaction of any type. It may be a single byte, or a complex transfer of data items.

Step 1: List the modes in which the system operates that result in different flow control and/or timing requirements.

Step 2: For each mode, determine all input/output requirements of the system. List all messages with their sizes, average, typical, and worst case periods.

Step 3: List the events that have special timing relationships. Generate a chart of events showing their relationships (Event Timing Relationships - ETR).

Step 4: For each message, indicate what event invokes generation of the message.

Step 5: Develop a test plan to detect and record events that meet the system requirements. (Thinking about how to test the software is an essential part of the design).

Step 6: For each message, indicate what function(s) is(are) required to process it.

Step 7: List all temporally related messages together. These are messages that can be processed sequentially without intervening delays. Distinguish between those that **can** be processed sequentially from those that **must** be processed sequentially.

Step 8: Allocate the processing of messages to tasks according to message priority and temporal relationship. Pay special attention to deadlines.

Step 9: Assign priorities to tasks, dictated by the priority of the messages in the task. Highest priority tasks will typically be associated with hardware interrupts. Insure that the hardware supports the interrupt priorities dictated by the design.

Step 10: Produce high level PDL indicating the functions called. Use packages to combine logically related objects and to provide isolation from those that are disjoint. Use functional decomposition to provide additional detail. Provide the greatest detail for those areas that are most likely to impact the performance of the system.

Step 11: Develop threads based on the individual paths within each task.

Step 12: Document assertions made in order to guarantee the required timing. For example: "High Priority Task A will only preempt Task B once per 20ms, and for a time less than 3ms in length." Maintain a data base of all assertions that are related to external interfaces.

Step 13: Produce static timing budget of each path segment, using "best estimates". Build a model based on threads and timing data to compute processor utilization.

Step 14: For execution time estimates that are uncertain, order them in terms of greatest risk.

Step 15: Starting at the top of the UNCERTAIN list, prototype each segment to determine actual time of execution.

Step 16: Feed actual times back into the timing budget and recalculate processor utilization with the model.

Step 17: Model event activity and compare against requirements.

Step 18: Develop a worst-case (or several worst-case) scenarios that can be used to test the system for meeting the difficult timing requirements. These should be used to convey the scope of the real-time concerns to the individuals writing the software code.

As new I/O and Events are added to the system (because of mid-course design changes or because of previous oversight), iterate on the above steps to maintain processor utilization and response time metrics. Using this information, intelligent decisions can be made regarding the efficiency of code required. The process steps above should be automated to the greatest extent possible so as to facilitate design changes, and to allow exploration of alternatives. A sample timing budget is shown in Appendix G.

The general philosophy should be to always have a spectrum of choices to select from that provide increasing performance, at the sacrifice of memory, complexity, and ease of maintenance. By knowing the difficulty of the real-time aspect, the program can be written to maximize maintainability while still meeting the performance objectives. A program that is maintainable but does not function because of performance problems is just as useless as a program that functions but is not maintainable. Both objectives are essential.

2.3 Testing

The design should incorporate testability as an essential requirement. It should be assumed that if the system can not be tested, then it does not work. Furthermore, mechanisms must be provided to allow logging of software faults whenever this is feasible. Experience has shown that real-time systems frequently fail under actual conditions, when they did not fail under nearly identical conditions in the lab. This necessitates being able to provide some useful system-state information at the point of failure so that some insight is provided to the maintenance engineers.

Testing should be done in a way that makes automatic regression testing possible. That is, command files should be established for the testing of each CSC (Computer Software Component) down to the unit level such that the suite of tests can be run in a fully automatic mode. Command files on the development host will place the simulator system into a predefined state, initiate the executable image on the target and collect the results of the test. When necessary, breakpoints may be set using command files for the debugger. Source lines that will require a breakpoint should have a test *point comment:* "--TPxxxx" where xxxx is a 4-digit decimal number identifying a unique test point. The automatic test procedures (command files) should reference test point numbers, and will be machine-translated to line numbers by a TEST_POINTS tool. At a minimum, each testable requirement specified in the Software Detailed Design Document (SDDD) should be verified in a test. Additional tests should be written as necessary to improve the reliability of the software.

2.4 Prototype Issues

During the development of the prototypes, several problems were encountered that required design decisions. The critical areas that required prototyping were enumerated as:

Graphics Display Updates

Communications Driver

Rocket Guidance Calculations

Operator Interface (Mouse)

2.4.1 Graphics

The graphics updates entailed placing target symbols, rocket symbols, and the targeting reticle [cross hairs] on the screen fast enough so that any perceptible motion was not lost due to processing overhead. With such a large number of targets and rockets moving, the number of screen updates is significant. Essentially, 1235 screen updates must be done every second, which allows a maximum of 810 microseconds per update (assuming 100% CPU utilization). An update consists of erasing and redrawing a symbol containing between 6 and 33 pixels. Initial software took advantage of standard software supplied by the graphics interface manufacturer to draw the symbols. Timing measurements indicated that it took well over a millisecond to draw a simple symbol. As a result, it was necessary to scrap the vendor supplied routines, and provide a more customized interface that provided better performance. Although the rockets travel fast enough so that they cross a pixel boundary (move) every 100ms, the targets often will not move every update (100ms). It is possible to significantly reduce the processing time by taking advantage of this fact, however it is still possible for all 100 targets to cross a pixel boundary during a single update period. This worst case must be taken into account. From an operator's point of view, it will not matter if some targets are deferred into the next update period, so if the software can ride through this "overload" condition for one update, it is possible to reduce the worst case processing requirement nearly in half.

The machine code statements are used to implement the Put_Pixel procedure. This procedure performs bit manipulations that could not be achieved with the Ada code generator, and were required for every pixel displayed (or erased) on the screen. Variable shifts were required to select the appropriate bit to set or clear. Although the same effect can be approximated with divide or multiply instructions, the execution time is one seventh as long when using the shift instruction.

2.4.2 Communications Driver

The communications driver must process at least 30 messages per second. After the design change to incorporate the Simulator into the BDS program, these communications occur as rendezvous operations. They are performed by the extended runtime that supports distributed Ada, and use buffer queues maintained within the runtime. Direct access to the network is not provided to the application program. If the simulator is redeveloped as a separate Ada program, it can either emulate the rendezvous protocol or the BDS can be modified to provide the communications with an Ada task serving as the interface driver. In this case, the low level interaction with the Ethernet hardware will be shared by the distributed runtime and the application program.

2.4.3 Rocket Guidance

Rocket guidance calculations proved to be quite time consuming. Initial attempts at using Ada fixed point calculations led to frustration due to the fact that on one compiler used, fixed point was being emulated using floating point. Fortunately the production compiler supported fixed point in a more efficient manner. Previous experience had provided warning that it was difficult to get fixed point numbers with a 'small that is not a power of two (most implementations restrict representation clauses on 4X'small to a power of two).

Therefore, it was imposed on the system design that the hardware provided all dimensions as a power of two value. For example, the METERS type used to provide battlefield position had a **delta** of 0.125.

Several application dependent optimizations were implemented to radically reduce computation time. For example, since accuracy became very important as the rocket approached the target, it was possible to take advantage of the fact that the target could not move significantly during the final ingress phase of flight. Normally the motion of the target must be accounted for in any targeting system, but since the BDS recomputed target position every 100ms, a simple forward extrapolation provided sufficient accuracy to guarantee a "hit". This replaced a more complex iterative algorithm which was used to compute the angle-to-intercept values.

Furthermore the tangent function was replaced with a lookup table, the arc tangent function with a rough approximation function, and the square root function was designed to run in acceptable worst-case time, with better accuracy at closer ranges.

2.4.4 Operator Interface

The mouse interface has turned out to be one of the more critical components of the system. Early tests indicated that poor response time on the mouse resulted in erratic movement of the targeting reticle. This made it nearly impossible to lock on a target and would result in mission failure in a target-rich environment. Another aspect about the mouse interface that makes it complex is the limited hardware support for the mouse. Only a single byte buffer is supplied for incoming data. This translates to a lost message if bytes are not read by the application software prior to the next byte reception (2 ms). An interrupt task is used to accept the data and buffer it for display processing. Care is required to prevent the high priority mouse task from being suspended while display processing was in progress.

The BDS source code is listed in appendix H along with the interface design documents and simulator specifications in appendices B through E. A graphical overview of the top level BDS design appears in Figure 1.

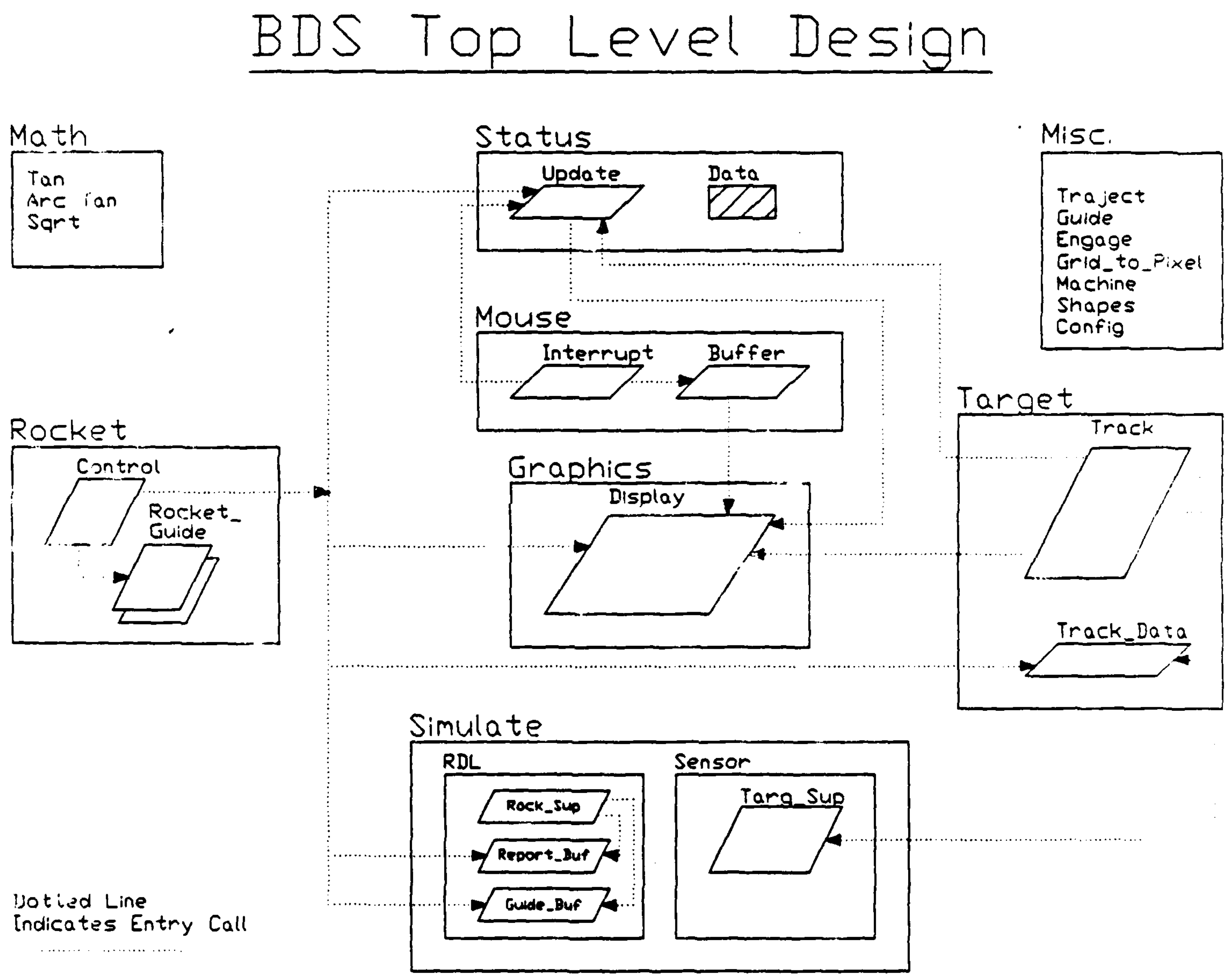

Figure 1. Top Level BDS Design

3. Compiler and Language Problems Encountered

Currently the "extended" features of Ada are not widely used, and therefore have the greatest number of latent anomalies. They should be carefully tested during integration to determine their reliability. Note that the vendor was contacted for a list of "known" anomalies, and they indicated that no such list was available. In fairness to the compiler vendor, their product is believed to be the most advanced Ada compiler of its kind. It is a comment on the complexity of real-time systems that there are still many problems with the Ada compilers. The runtime executives are attempting to solve many of the issues of real-time programming and need to be extremely robust, yet fast. These are two aspects that generally work against each other.

1) LONG_FIXED division was unreliable. Certain numbers (resulting in bit patterns very close to 1FFFFh) caused divide error. This manifested *itself* by *causing* a NUMERIC_ERROR after hours of operation and hundreds of rocket launches and target intercepts.

2) Improper Inlining of code statements. If the last instruction of the calling sequence used the same register as the first instruction of the machine code procedure to be inlined, the code generator would exchange the two instructions:

```
        mov   [bp-10],cx
        mov   cx,[bp-20]        -- code statement begin
          ...

would become:
        mov   cx,[bp-20]
        mov   [bp-10],cx        -- reorder, resulted in storing of incorrect value
```

Note that this problem appeared after the code in question had already undergone successful integration testing. It appeared suddenly after a new re-compilation caused different registers to be used (with no changes in compiler switches).

3) If an interrupt occurs when the auto-increment direction flag is set, it is not restored properly if the interrupt is handled by an Ada task (using the INTERRUPT_HANDLER **pragma**) which makes an entry on another task. This only appeared when the system was heavily loaded, and periodic rates were set to the full speed. Fortunately the one place in the code where the direction flag is set (normally it is cleared) was executed quite frequently so the problem could be reproduced, although not in a predictable fashion.

4) Complex expressions did not generate the correct code sequence. Actual parameters containing array aggregates, which in turn consisted of multidimensional array references with non-integer subscripts resulted in a failure for the appropriate segment register to be loaded correctly. The graphics task takes a parameter list consisting of the old and new positions of an object(x,y), the object type (rocket,target, etc.), and a color. To determine the color, an array indexed by the object type in one dimension and a status flag indicating if it was engaged for intercept was used in the other dimension. This causes targets to "light up" when engaged for intercept. However, it did not work and the code was rewritten to create temporaries during each step of the expression. The failure mode was to select the zero(0) color - black, which gave the appearance that nothing was working, when in fact invisible targets were moving on the screen.

5) The **pragma** to establish task storage size did not function. This resulted in the program terminating before initial elaboration was complete. Basically the program would simply crash with no exception trace back (due to the fact that the program had not completed elaboration). This required a programmer to single-step through the code to find where the problem was. The solution was to use a linker option to set the library stack size, which did work although it applied the same stack size for all library tasks. A general comment is that it is extremely difficult to determine the correct size required for a task stack. Ideally an

automatic approach to setting the stacks would be supported. It might use pragmas to provide maximum call depth information to the compiler. The compiler could compute the worst case storage requirement for each task type and allocate their stacks accordingly.

6) Deeply nested exceptions did not propagate properly. In a task, a loop with an exception block called a procedure which had an exception and no exception handler. The exception should have propagated to the exception block within the loop (which had an "others" clause), but instead propagated to the outermost level of the task, causing it to terminate silently. Debug trace statements were placed in each of the tasks but these were not invoked when exceptions occurred. The result was either a deadlock, or other tasks getting TASKING_ERROR when rendezvous were attempted.

7) **Package** Calendar elaboration check was not performed properly. A number of problems with elaboration were encountered (due to library tasks starting execution before other units are elaborated). One strange problem was due to the fact that no elaboration check was performed prior to calling the Calendar.Clock **function** to establish the periodic start point. Apparently the Calendar **package** body had not been elaborated and the Clock **function** returned the time of a few hundred microseconds into mission time. Then after the calling task was suspended for a rendezvous, the Calendar **package** was elaborated, which set the TIME value to some time in 1987 (a very large number). When the task resumed execution, it reached the end of its loop and attempted to compute the **delay** necessary to achieve the desired interval. Since the delta time was almost 2000 years, it exceeded the range of duration and a NUMERIC_ERROR was raised (although a TIME_ERROR) should have been raised.

8) Calling more than a single **entry** from within an interrupt task (using **pragma** INTERRUPT_HANDLER) can result in a system crash. Using the rendezvous to signal a

background task to continue execution is essential in real-time applications. This mechanism may not be reliable on all implementations and special precautions are necessary for its use.

9) Named associations in aggregates generated substantial amounts of additional runtime code over using simple positional associations. This was true even if the values were positionally correct as well. This resulted in having to limit the named associations and to put the names in comments following the statements. This had a negative impact on the readability of the program.

The impact of having many failures in the runtime and generated code is demoralizing on the engineering staff. It becomes apparent that the most difficult problems to find are those of the runtime and generated code, since one expects the Ada statements to work as specified. There is a realization that the success of the project is out of your hands, and in the hands of the compiler vendor. No matter how good the engineering team is, the system will not work if the translation from source to the generated code is incorrect. This is unusual for real-time programmers who are familiar with assembly language, where there are far fewer discrepancies. When discrepancies do exist, they are much easier to find.

Finally, a problem that surfaced with the Ada language was a standard way to provide atomic transactions that could cross the application code to runtime code boundary. The application has a requirement to accept frequent interrupts, buffer the data to a certain point (based on the input stream), and then perform a considerable amount of processing on the data, while new data is arriving. This is done by having an interrupt task perform the buffering, then passing the data off to a background task. The difficulty is that the interrupt task may not be suspended (for any reason other than to service higher priority hardware interrupts). This implies that a conditional rendezvous is required. However, what is really required is the ability to queue the buffer and request, then signal the background task if it is suspended waiting for new data. Essentially there are two approaches to this problem: 1) provide a sufficient number of buffer tasks so that they can act as surrogates on the **entry** queue of the background task, or 2) maintain a flag that is only set when the buffer task is ready to immediately accept a rendezvous. This requires that the background task disable any type of preemption; check if there is more work to do; and if not, perform the **accept** statement. Presumably the runtime will then allow preemption only after placing the background task in a position to immediately accept the rendezvous. The interrupt task obviously will not attempt a rendezvous with the background task unless the flag is set. Both of these solutions have serious drawbacks. The surrogate task approach requires substantial optimization on the part of the compiler and runtime. Furthermore, it may obfuscate the intent of the rendezvous. The second approach is very implementation dependent, and is prone to error if used by other than very experienced and careful programmers. What is clearly needed is an asynchronous form of task communication. This could return to the language as revised forms of the "SEMAPHORE" and "SIGNAL" generic tasks defined in the "Preliminary Ada Reference Manual" [3] . Although an argument can be made that implementations can provide the same effect as these predefined task types, to depend on implementation optimizations for such crucial real-time operations is a questionable design approach.

4. Hardware Problems Encountered

Several hardware problems were encountered during the development of the BDS system. For the most part, they presented minor nuisances, but in some cases resulted in days of additional testing to isolate the problem.

One of the five Ethernet cards used arrived non-functioning. After approximately one-half hour of performing various tests, the failure was isolated to a bad "station address PROM". This nonvolatile memory holds a unique network address for communication on the Ethernet. Upon very close inspection of the PROM integrated circuit, it was noticed that one of the pins had been bent under the chip. By removing the chip, straightening the bent pin, and reinserting the chip, the problem was solved in a few hours.

The 80386 computer card has a VLSI chip (80C206) used to control interrupts, timers, and several other features in the computer. One feature that was used to obtain accurate clock values was a mode which latches the two bytes of the 16-bit counter as well as a status register simultaneously. This is necessary because the counter is changing value every 418 nanoseconds and getting reliable values is difficult without such a latch. Unfortunately, the latch did not work. The programming of these chips is extremely complex, and the first assumption when something fails is to suspect the software that interfaces to the chip. The documentation was rather vague, and as a result approximately two days were lost until it was concluded that the chip may be bad. Note that the timers worked fine, and the rest of the chip's thousands of transistors apparently worked fine as well, so hardware failure was not suspected. The only failure symptom was that once in a great while, a time would appear to jump backward by a small decrement. This was due to reading one byte while the status of the other bytes changed. By trying the exact same software on another computer, it was determined that there was indeed at least one bad gate on the 80C206 chip. A replacement chip was ordered and when it arrived, it did not work at all. A month later the second replacement chip did work, and solved the timing problems.

Although not strictly a hardware failure, the documentation describing the Ethernet hardware had numerous errors. Weeks were spent trying different initialization sequences in an attempt to debug the network drivers. For example, the board has an on-board network transceiver, and may alternatively use an externally supplied transceiver. The bit that controls which transceiver is used was incorrectly specified in the documentation. It was pretty much an act of desperation that allowed the error to be found.

Likewise, the technical manual on the mouse, incorrectly specified jumper settings which establish the communication protocol between the mouse's processor and the BDS processor. Once again, it was trial and error until something worked, and the error condition could be verified.

5. Distributed Ada

Note that while multiprocessing may be used to help solve the performance problem associated with Ada, it is not a cure-all. Developing software using inefficient algorithms and then trying to apply a large number of processors in order to get the desired performance is not suggested as an appropriate design method. Suitable algorithms can make a much more significant impact on performance than adding several processors. The additional processors should be regarded as a fine adjustment on the performance rather than a simple multiplier. This is dictated by the fact that the recurring cost of hardware, while continuing to fall on a cost/performance basis, is still not inconsequential. While adding another processor cannot compensate for poor software, it can more than compensate for some efficiency lost because of immature compiler technology. If the high level language can take the complexity out of the distribution effort, then the result is a system that will perform better and cost less over the life of the system.

5.1 Definition of Terms

The term "Distributed Processing" is frequently used, and often with meanings that imply radically different architectures. For this reason, the use of the term within this paper is explicitly stated.

Distributed Processing, for the purposes of this project, shall be defined as: "a multiprocessor system characterized by having communicating autonomous processors, where the communication response and throughput are significantly lower than local memory access".

"Significantly" in the above sentence implies a difference that is greater than an order of magnitude. The essential aspect of this definition is the recognition that program performance of these systems may be radically altered by changes in configuration. In particular, when functions that interact within a processor become separated onto two or more processors, the communication overhead can be dramatic. Therefore, in general, related functions separated by a processor boundary tend to be loosely coupled with each other.

5.2 Project Objectives

One of the principle project objectives is to determine how Ada tasking can be used on a distributed system to improve total system performance. There are many advantages to using the Ada tasking model as the sole abstraction of concurrency. Among them are:

1) The ability of the compiler to check interfaces between physical processors.

2) A consistent approach to parallelism - all concurrent activities are expressly stated with a consistent formal mechanism making the system less complex.

3) Re-configuration is facilitated, since the interface between communicating tasks on a processor is the same as that among separate processors, thus allowing tasks to be migrated more easily.

4) Consistency helps to make distributed testing and debugging more easily supported by compiler implementers. Ad hoc approaches make debugging tools prohibitively expensive.

Several studies have implied that the synchronous rendezvous required by Ada is not practical for real-time distributed systems [report to Real Time workshop at IDA, July 1988]. This conclusion is drawn because of the communication required by the Ada semantics creates tremendous overhead in the network-based systems studied. Typical round-trip communication times were measured at 20ms for a single message/acknowledge, of which several might be required for Ada's rendezvous semantics. These numbers were presented as being processor, operating system, and network independent. The work of this project implies that the message transfer times are distorted in many of these studies by the unnecessary overhead associated with message presentation to the network. In every case, an intervening operating system such as UNIX[1] or iRMX86 [2] is used rather than direct interaction of the runtime with the network interface. These operating systems do not provide high performance pathways for user programs to access network facilities. Overhead is incurred by requiring a context switch between User and Privileged mode, and often a full process switch for every message transmission/reception.

Furthermore, additional network protocols such as TCP/IP are used, and packets are often manipulated by a "smart" network interface card which results in actually slowing down the communication. Studies done on a previous project indicated that it is possible (in a lightly loaded Ethernet system with commercially available hardware) to perform the following steps in under two (2) milliseconds:

[1]UNIX is a trademark of AT&T

[2]iRMX86 is a trademark of Intel Corp.

1) call a procedure requesting a block (512 bytes) of data,

2) generate a packet with the request for data,

3) transmit the request across the network to a server processor,

4) server processor interprets the request (disk read),

5) read the data from the disk (1ms),

6) transfer the data back to the requester,

7) the requester accepts the data, and is ready to issue another request.

Since a 2:1 interleave was used on the disk, the effect was to achieve the same data rate from the remote disk as what was available on a local disk (identical hardware). This is due to the fact that with a 2:1 interleave, there is a millisecond delay (actually 980 usec) between reading one disk sector and the next. The disk is formatted this way to allow the operating system enough time to read the next sector before it is missed and a full rotation is required. The developed software was able to perform the two way communication across the network during the 980 microsecond inter-sector delay with enough time left over to setup for the next operation. The software overhead associated with the network was under 380 microseconds, since the actual transfer time on the network for a 512-byte packet and a 64-byte packet was approximately 600 microseconds.

The point in the discussion above is that people may be rejecting distributed Ada unjustly. Similar to the indictments of Ada that came about with initial releases of compilers, the initial studies of distributed Ada are based on prototype implementations and are built in a way that is unlikely to be used in a real-time system. It is essential to accurately determine what the penalties of a synchronous protocol are prior to rejecting the Ada rendezvous model as being unsuitable. Once the requirements imposed by Ada are clearly understood, it may be possible to support these requirements more efficiently in hardware, and thereby nearly eliminate any penalty associated with using the Ada tasking model.

One point that cannot be ignored is the substantial difference in overhead associated with various tasking combinations. In particular, the presence of an **abort**, timed entry, or selective wait with either a **terminate** alternative, or a **delay** alternative greatly complicate runtime processing. In addition, the reliability of the network (transmissions are not guaranteed) also increases overhead substantially. This project has shown that a simple rendezvous in an error-free environment is achievable on commercially available hardware in under 1ms.

5.2.1 Unit of Distribution

Parallel systems research often concentrates on how to achieve parallel computation of a process that is described in a serial language. Great effort is applied to detecting independent calculations and executing them on separate processors. Claims are made that the program design must be fundamentally changed in order to achieve good performance on different parallel architectures. Our findings are that, while this is true for the general class of computer programs, it is not so typical of embedded applications.

Embedded applications, because of their interaction with several real-world events, tend to have natural parallel threads of control. In fact, programmers of such systems often try to artificially reduce the number of parallel threads in their programs because of the overhead associated with switching between threads. In these cases, they combine two or more threads. For example, in an aircraft computer where a response to pilot commands must be processed within 50ms, and new inertial data is received every 40ms, the program might check for a new pilot command every time the inertial data is received. This combination of threads insures that the pilot is monitored frequently enough to meet specification. The result is that the code has been convoluted by the need to achieve a "low-cost" context switch. Furthermore, the pilot input is being checked more frequently then necessary, and yet will still incur an average 20ms delay. Ideally a separate task would be created for both functions, and they would only execute upon (interrupt from) receipt of pilot command or inertial data. However, the context switch associated with the ideal approach may cause the program to miss deadlines in the presence of a large number of pilot commands.

With the latest generation of Ada compilers, this context switch overhead is being substantially reduced, and therefore it is practical to express "naturally" parallel activities as Ada tasks. Since the expression of the dependencies of such tasks (parallel threads) are clearly indicated by **entry** calls and **accept** statements, the program can be distributed in a straightforward way, provided that the semantics of the Ada tasking model are faithfully preserved for Ada rendezvous and shared variables.

6. Distribution Implementation Details

The Distributed runtime code is composed of five modules: the Rendezvous Services, Linkage Code, Network I/O, Network Setup, and a small Vendor Runtime Interface.

The Distributed Rendezvous Services provided the initialization, remote elaboration start, remote entry, local entry, selective wait (among either local or remote tasks), remote start accept, remote end accept, and local end accept. Additionally, routines were provided that execute at interrupt level to respond to incoming messages. These include: begin elaborate, end elaborate, request entry, and accept complete.

The Linkage Code provided additional information such as the length of parameters for the distributed rendezvous calls that is not normally needed for tasks that share memory.

The Network I/O and Setup modules provided direct interface to the Ethernet hardware. All protocol management and packet construction was performed by this software, as well as direct initiation of packet transmission - and interrupt response upon completion of transmission or packet reception. Buffers were allocated and deallocated from a fixed size buffer pool for efficiency. For example, buffer deallocation time was under 5 microseconds.

Packets contained the normal 48-bit destination and source addresses, a field termed "receive control pointer", packet priority, sequence number, and total packet length followed by parameter data. The receive control pointer (RCP) was used by the receiving processor to determine how to process an incoming message, which task and/or entry is referenced, and to locate the proper reply message header. Use of these pointers reduced the amount of information necessary to be transmitted, and provided quick access to statically created packet headers.

The Vendor Runtime Interface module provided addresses for standard P and V semaphore operations used to "wait" and "signal" task execution.

The distributed runtime was developed using assembly language to be compatible with the vendor's runtime, and to more accurately determine what could be achieved rather than allow possible high level language inefficiencies from biasing the results. Future work would probably benefit from an Ada implementation, and it would be interesting to compare the relative performance of the two approaches. Refer to Appendix I for source code of the distributed runtime code.

7. Distribution Issues

7.1 Language Issues

The Ada Standard provides a degree of confusion as to what semantics are required in distributed systems. Examples and possible solutions for these situations are:

1) Timed **Entry** Calls - The Ada language reference manual (RM) implies that the rendezvous will be cancelled if it cannot begin prior to the **delay** specified. This leads to confusion on a distributed system where the **delay** may expire prior to being able to request a rendezvous with a remote task. But the Ada semantics would have permitted the same rendezvous to occur if the **delay** were 0.0 or negative, since in this case the semantics of the conditional entry call are used. It is pretty clear that the desired semantics are to ALWAYS attempt the rendezvous, regardless of the **delay** specified (like a conditional call), and if the server is not at a corresponding **accept**, then wait the specified **delay** prior to cancelling the call.

2) **Package** Calendar - Although no explicit confusion exists, programmers may be misled by the simplicity of the discussion on TIME. The time of day is expressed as the number of seconds (presumably *since midnight local time*). This is contrary to many embedded systems which operate on Mission Time or Universal Time (Zulu), although this is still possible with the current definition of Ada. What is perhaps missing is the ability to update the system value of time. This problem is especially acute in distributed systems since they frequently operate from independent oscillators. This results in synchronization being required at initialization and periodically thereafter due to differences in clock rates.

Application software within tasks frequently time-tag information indicating when the data was sampled from the external device. If this information is then transferred to a remote task operating on a separate clock, the interpretation of the time-tag is likely to be faulty, resulting in error or even system failure. Time is further complicated by the need to synchronize, because it will result in some of the clocks being forced to jump forward or even backward in time. This noncontinuous aspect of time makes synchronization in distributed systems very complex. It is suggested that **package** Calendar be expanded to support applications modifying the system value for time. Clear semantics must be specified, especially with respect to not impacting any delays currently pending. Ideally, discontinuity in time should be allowed to "creep back on schedule". That is, an epsilon time would be provided to the runtime system, which would gradually advance/retard the clock at a specified rate until the epsilon was eliminated. For example. if the clock was determined to be 500 microseconds behind real time, an epsilon of 0.0005 would be specified with an adjust rate of 0.001. This would imply that the clock would be accelerated at a rate of 0.1% (1 micro second per millisecond) until it was on "real" time. This prevents significant errors during computation of elapsed time. Another approach is to have each processor maintain a continuous increasing mission clock which is used to compute elapsed times and to translate mission time to time-of-day whenever necessary. In this type of system, each processor keeps an adjustment of time to correct differences between any other processor with which it communicates. Obviously a common clock stream going to each processor is a preferable solution to any of the above if the system architecture supports it.

3) A Failed Processor Node - The principle designers of the Ada language have stated that the Ada language does not specify what happens in the event of a hardware (CPU or memory) failure. This makes tremendous sense from the point of view that no set of instructions can have meaningful semantics if the underlying hardware is unable to execute the instructions correctly. However, when a program is distributed over multiple processors, and one fails, the integrity of the other processors is not necessarily compromised. They could conceivably continue to function correctly and it might be appropriate to know the semantics of any interaction with tasks or data on that failed processor. It is suggested that the semantics of such a failed processor node be identical to those of a task which has an unhandled exception. That is, the task will go to completion. Any clients in a rendezvous will have TASKING_ERROR raised at the point of call. For shared data on that node, a

new exception should be defined, such as ACCESS_ERROR, which will be raised when any access to storage fails. Note that this is different than STORAGE_ERROR in that the latter is only raised when available storage is exceeded. Recovery from the two types of faults is likely to be different. The ACCESS_ERROR might also be appropriate for memory errors on the local node as well. Finally, in fault tolerant applications where a node may come back on-line after a failure, ACCESS_ERROR might be a more general approach to reporting an error during rendezvous than TASKING_ERROR. This would allow a node which was intermittently accessible due to communications problems to continue to operate rather than require otherwise healthy tasks to become completed. In these situations referencing the 'TERMINATED and 'CALLABLE attributes while the node executing the designated task is unreachable would also raise ACCESS_ERROR.

4) Package System - prohibits two separate representations of a system dependent type within the same program. This causes difficulty in distributing a single program among heterogeneous processors. A solution might be to create a function "Processor" which returns a static enumeration type (Processor_Type) indicating the processor the program is actually executing on. Each of the named numbers in package system could be changed to be a function which returns a static value and takes a single parameter to select the processor. The default parameter for the function would of course be the "Processor" function. Types ADDRESS and NAME could similarly be referenced by ADDRESS'CPU(Processor_Type) which returns the type of ADDRESS on the specified processor. Once again the normal types for ADDRESS and NAME would be preserved and would refer to the executing processor. [This solution may need tightening up.]

5) Priority preservation by the network is an issue that should be addressed in future language revisions. The term "available processing resources" in RM 9.8(3) can also include

the network, since it may be managed by the Ada runtime. The RM should clarify if it is acceptable to have a high priority task blocked waiting for low priority tasks to perform network transfers. The preferred approach is to require implementations that support priorities to also have the priorities apply to network scheduling as well.

7.2 Issues Addressed By the Demonstration Project

Essentially every paragraph of the RM Chapter 9 (Tasking) has an impact on distributing Ada programs as Ada tasks. Achieving predictable timing in a distributed Ada application is also a major concern. This implies the ability to maintain task priorities in network transactions. Most commercially available network hardware does not support preemption of message traffic to higher priority messages. This results in a high priority task being blocked while lower priority tasks transfer data (sometimes referred to as Priority Inversion). Special care must be taken to avoid this effect and to guarantee response time to high priority tasks.

The issue of shared memory has frequently been addressed by totally restricting its use. Although the demonstration project did not need distributed shared variables for the initial prototype, their availability would give greater generality to what can reasonably be distributed. Some analysis was done however to determine what might be needed to support such variables. The approach that would be used to implement distributed shared variables would be to assign them to segments which could be mapped as "not present" by the memory management system in the processor. The segment descriptor tables are then programmed to recognize segments of "Network Data", and if the data is not resident, result in a page fault. This page fault results in suspension of the task and a network message requesting the data being multi cast to all processors which access that segment. The node having the data marks its segment as being not resident and transmits the data segment to the requester.

Upon receipt of the data, the requester marks its ~~segment~~ as resident and makes ready the suspended task for execution (possibly preempting a lower priority task). It is essentially a complicated version of virtual memory, utilizing the virtual memory support provided by the 80386 processor. Several types of Network Data are envisioned: Simple Static data that does not migrate; Migrate-able Data as described above; and Replicated Data that is available locally in read form on all processors, but can be written only by a single master, resulting in a broadcast to all interested nodes.

7.3 Deferred Issues

The following issues deserve immediate attention but are well beyond the scope of this project.

Load Management - dynamic task migration to provide the desired loading

Fault Tolerance - Detection, Isolation, Roll Back and Recovery, etc.

Design Issues - Paradigms for a "good" distributed design

Analysis of Distributed Behavior - Predicting overload conditions, deadlock, etc.

Limiting the Impact of Changes in Distributed Systems - how to compensate for different performance because of configuration.

Incremental Upgrades While On-Line (Never-Stop) - dynamic binding of Ada name space. Maintaining system state in the presence of "new" objects.

Debugging - synchronous halting of all processors and system clocks. Providing "transparent" network debugging support.

Heterogeneity - common interchange formats, negotiation of desired formats between processors.

Security Barriers - Maintaining multi-level security among secure processors. Data labeling, real-time encryption, trusted network software.

Network Interface Standards - What impact does "required" compliance with OSI (Open Systems Interconnect), MAP (Manufacturing Automation Protocol), or GOSIP (Government OSI Profile) have on real-time systems? Is compliance desirable?

7.4 Architecture Support

In order to achieve optimal support for distributed Ada applications, new architectures will be necessary that closely relate to the Ada semantics. Although this topic is also beyond the scope of the project, several ideas regarding ideal architectures have come to light while performing work on the project, and are partially documented here.

For processors that can be connected via cables, an ACTIVE STAR network may provide good solutions to many of the distributed Ada problems. Since there is a need to provide a common sense of time among distributed systems, the star hub could be continuously transmitting data to and from each of the nodes. This data stream would serve several functions. The first function would be to supply a common and constant clock stream to each of the nodes. The second function would be to provide constant status on the health of each node. If a node fails, a watch dog timeout on the interface card would trip and halt the data stream to the hub. This would result in a "Failed" status at the hub. Obviously the data streams would also serve to transmit data to and from the hub and other nodes. Control information embedded in the stream would indicate what should be done with the data.

The hub design is absolutely critical. To leave out functionality would result in significant performance degradation. As a minimum, the hub should support the following capabilities:

1) It should be able to be replicated to provide N-version fault tolerance.

2) It should preserve the priority of the task requesting the transfer for all network transactions.

3) Network Data Store (NDS) should be provided to cache shared variables and to hold the necessary data to resolve inter-processor tasking states.

4) The NDS memory should be high speed (under 35ns) and should be time division multiplexed among all nodes, simulating a multi-port memory.

5) Access to the NDS memory should include primitives for indivisible operations such as test and set, and atomic ADD and QUEUE operations.

6) Security provisions should be included to electrically prevent data labeled as classified from being transferred to a node of lower classification.

7) Node interfaces to the network should appear as memory in the processor address space. Task control blocks for tasks which have remote communications should be located in this memory, which is maintained to be consistent with the image at the hub.

8) Node interfaces should contain a 64-bit mission clock that can be reset simultaneously with all other nodes, stopped simultaneously with all other nodes, and is clocked simultaneously with all other nodes. In addition, the interface card would have a 32-bit 40.233usec clock which would be used as a DURATION mapped delay device. Rather than interrupting the CPU at a regular period, the timer would be programmed with the shortest delay currently pending. This would provide very accurate delay timeouts, without incurring overhead when delays are not being set. To eliminate the software overhead associated with determining the proper value for the timer, custom hardware could be supplied that would accept a delay value and a task ID. The hardware would support up to 128 delays directly and allow software to resolve any delays beyond 128 (at most one "true" delay need be set by any task). The supplied delay would be added to the 64-bit clock and an absolute time saved in a register file sorted according to increasing time. If the newly requested delay is earlier than the earliest delay set, its time would be subtracted from the 64-bit clock and the resulting value placed in the count down timer. As times expired, the earliest time would always be placed in the timer. The task ID would serve to notify the runtime as to what delay timed out (along with an interrupt on a programmable level), and to allow for cancelling of the delay. All of the clock circuitry could easily be supported within a current technology gate array.

9) Built in Test provisions should be included in the hub.

10) Debugging support, including simultaneous halting of all nodes (via non-maskable interrupts from interface card) should be an option for hubs. The halt would also serve to disable counting in all node clocks so that perceived real-time is uninterrupted. Obvious features such as data logging of "meaningful" data should be conducted at the hub.

11) Provisions to provide network manager control from one or more privileged nodes. This might include getting network status, forcing a clock synchronization with a new mission time, broadcasting data, or shutting off other (wayward) nodes.

Such hub designs are believed to be practical with current technology for reasonable sized networks (under 32 nodes, however each node could be a cluster of processors or a gateway to a hierarchy of other nodes). A typical medium might be fiber optics for long haul applications or coax cable for shorter runs. Data rates of 100Mb/s could easily be achieved using 125MHz modulation with 5B/6B codes. The effective throughput should exceed 12MB/sec with propagation delay times typically under a microsecond in the absence of contention.

8. Timing Analysis

Timing information was collected for several purposes. First, to insure that the program was operating correctly and to help isolate the source of timing problems. Timing information was also essential to evaluate the effect of optimizations. Finally, accurate timing information was necessary to compare the relative performance between the single and multiprocessor versions of the application.

Timing information was obtained with three techniques. The first technique used a preprocessor to insert procedure calls to a time stamping routine. These were inserted throughout the program at every procedure entry and exit point, as well as task rendezvous points. In addition, the runtime was modified to make a call to the time stamper whenever a context switch occurs. This made it possible to recognize when a task is preempted, and avoided incorrect charges of execution time. An example listing resulting from a time stamp execution trace appears in Appendix G.

The time stamp mechanism places timing information as well as a unique identifier in a circular buffer during execution. This buffer was set at 64K bytes and used 8 bytes per time stamp. It "wraps" every 8192 time stamps, overwriting previous time stamps. This allowed a trace back of several 100ms cycles through the program from any stopping point. It was possible to edit the pre-processed source code to selectively enable/disable time stamps which extended the trace back up to several minutes. Additionally, it was a simple matter to disable all time stamping by replacing the stamping procedure with a "return" instruction. Since the time stamp procedure requires slightly over 20 microseconds, using it did intrude on program execution. Often only one or two time stamps were enabled to provide accurate timings without significant perturbation of the program. The clock used had a resolution of 419 nanoseconds and is software extended to 33 bits.

The second approach to timing was used to measure aggregate processor utilization. This technique involved use of custom hardware that permitted starting and stopping one or more processors simultaneously. Essentially, a separate communication signal was configured that could invoke a non-maskable interrupt (NMI) in each of the processors. This line was activated by a dedicated "timing" processor which would initially signal the processors being timed to start, then signal again, to stop the processors. Each processor would increment a counter within a tight loop during their "idle" time. The value in the 32-bit counter would provide a measure of the amount of time spent in the "idle" loop. As processor saturation occurred, the counter would not increase beyond those counts that had accumulated during initialization (prior to reaching a steady-state load). After close analysis, it was determined that the "idle" loop value is not valuable in the distributed system, because this is where the processor waits for communication. The "idle" time therefore gave a false impression that deadlines could be met.

The best indication of meeting deadlines was the third method of timing. To support periodic processing, each task computes the amount of time necessary to delay before the beginning of the next period. The durations computed were saved in a buffer using an approach similar to the time stamping technique. The most time critical task: "ROCKET.CONTROL" was monitored in this way to record how close it came to the next deadline. If the duration became negative, it was a clear indication of an overrun condition. That is, the time for the next period had already passed.

Times were collected for single and distributed processor configurations and are reported in Figure 2. The distributed system has two processors connected by a 10 Megabit per second Ethernet link. The results clearly indicate that the remote rendezvous can be completed in a reasonable time, even with large transfers of data required for **in out** parameters; and that there is a substantial performance benefit to distributing the program. This scenario with the distributed architecture allows 12 airborne rockets to be processed before deadlines are missed, while the single processor architecture can only support 7 rockets without deadline overrun. This is indicated by the point at which the available time prior to the next period becomes zero.

Figure 3 shows the times in microseconds required to perform the actual intertask communication using the distributed runtime. Note that the overhead roughly approximates a fixed portion of 175us and a data transfer of 1 byte every 4.03us.

	NUMBER OF ROCKETS				
	0	5	10	15	20
PROCESSOR					
Single CPU	97ms	31ms	-37ms	-87ms	-102ms
Distributed	95ms	61ms	27ms	-41ms	-69ms

Figure 2. Available Time prior to 100ms Deadline for Rocket.Control (25 Targets)

Remote Rendezvous	Microseconds
no parameters	490
1 out parameter 202 bytes	990
1 out parameter 1002 bytes	4214

For the 1002 byte rendezvous, the breakdown is as follows:

Message setup/transmit and context switch	110
total distributed runtime execution including round-trip message transfer and second context switch	3737
Out parameter copy after reception (end rendezvous)	367

Figure 3. Rendezvous Timings

9. Principle Findings

The Ada rendezvous model is practical (although not necessarily ideal) for distributed communication. Unconditional rendezvous with small parameter lists can be achieved with off-the-shelf communication hardware in under 1 ms. Although this is significantly higher than the 100us required for local rendezvous, it is still acceptable for many applications. More complex rendezvous mechanisms such as timed **entry** calls and selective waits with **delay** alternatives impose substantial additional overhead. As with non-distributed applications, the synchronous nature of the Ada rendezvous imposes additional task constructs in order to "uncouple" many inter-task communications.

Although Ada compilers now are near to being "full" implementations of the language, the most complex features are not yet reliable enough for life-critical applications. Errors in runtime code are still too prevalent. These errors are extremely difficult to detect because they only occur during particular coincidences of events. A general example is the execution of some particular sequence of instructions at the exact moment an external event invokes a context switch. This sequence utilizes an infrequently referenced flag within the processor. If the state of the task is not fully restored after the context switch, the particular sequence may fail to operate correctly. This type of error may go undetected after years of operation, only to result in total system failure at a particular instant.

The execution rate of both generated code and the runtime code is considerably better than that of compilers of 1986. However, checking code remains verbose. This will tempt real-time application developers to suppress the checks, which has a consequence of taking different paths through the code generator. Since these paths may not have been tested as thoroughly as the primary path (generating checks), the resulting code is likely to be less reliable.

The speed improvement of distributed Ada is not necessarily scalable. Although the parallel nature of embedded applications make them ideal for multiple processors, the individuals tasks are not usually balanced in processor loading. On a shared memory multiprocessor, scheduling can occur on a "next available processor" basis but this is usually not practical on a distributed system due to the locality of data. The "vectorized task" is a partial solution to this problem. To implement the guidance operation for up to 20 simultaneous rocket trajectories, an array of tasks was used. The actual size of the array was controlled by a configuration parameter. Each task in the array was passed a list of rockets to guide. If additional processors became available, the size of the array could be increased and the tasks could be distributed. The size of the individual WORK lists for each task would decrease correspondingly. This achieves a "near scalable" performance increase as processors are added.

Techniques for achieving distributed Ada via pre-processing the source code, or post-processing the generated code/runtime are acceptable for research, but unlikely to be so for any production environment. What is required is an integrated compiler/linker/tester that supports distribution of programs. A flexible compilation system would support several different levels of distribution. The same compiler could target a single processor, a shared-memory multiprocessor, or a physically distributed network of nodes, consisting of either single or multiprocessors. This would give substantial reconfiguration capability to system designers. Support for heterogeneous processors would additionally allow customization of the configuration to provide the best match of processing resources for application requirements.

Some aspects of the Ada language definition are silent about what should happen in a distributed system. For example, if a node fails, should future rendezvous to a task in that node get TASKING_ERROR or simply deadlock? What about a rendezvous already in progress with a failed node? What if the node fails, but then returns to service? These are all likely scenarios in typical distributed systems. A clear statement about what can be expected in these situations (or possibly control over what happens via **pragmas**) is necessary in future language revisions. Another area is the interpretation of the timed **entry** call. If the **delay** duration is > 0.0 and yet the **delay** expires prior to a message being sent to the remote task, should the rendezvous be terminated even if the accepting task is ready for an "immediate" rendezvous? Refer back to section 7.1 "Language Issues" for more detail and other areas of concern.

A software manager should not use Ada on a serious real-time project without source code to the runtime system. This is not for the purposes of modifying it, but rather to understand the detailed execution when necessary. This information is not available in even the best vendor documentation (which is sometimes incorrect anyway) but can only be verified by examining the source of the runtime.

10. Summary

This report discusses the advantages and issues associated with using the Ada tasking model for real-time distributed systems. A case is made for reducing the overhead of the remote rendezvous (and **shared** variable access) so that its use becomes practical. Some issues regarding the clarity of the Ada Standard are presented and possible interpretations are suggested. Several open issues are listed as areas that require further study. These issues such as fault tolerance, security, and dynamic re-configuration are extremely complex taken one at a time, and become even more complex in any combination.

Demonstration Project Final Report

A real-time demonstration project is described and was used to evaluate the effectiveness of the distributed Ada tasking approach. It also served to shed light on other implementation issues with Ada not associated with distribution. The demonstration software should prove helpful to other projects wanting to evaluate real-time performance, or to evaluate the capabilities of compilation systems with respect to real-time embedded applications.

11 References

[1] ANSI/MIL-STD-1815A-1983, "Reference Manual for the Ada Programming Language", American National Standards Institute, Inc., 1983.

[2] "Software Engineering Issues on Ada Technology Insertion for Real-time Embedded Systems", September 30, 1987, Final Report, Center for Software Engineering, CECOM, Ft. Monmouth, New Jersey; prepared by LabTek Corporation, Woodbridge, CT.

[3] "Preliminary Ada Reference Manual", ACM SIGPLAN Notices, Volume 14, Number 6, June 1979 (Part A).

12. Appendix A - Specification for the Border Defense System

The following document contains a description of a hypothetical battle management system used to defend a border. The purpose of this description is to act as a low level system specification for the development of software to implement such a system. This software will be used to assess the feasibility of using distributed Ada tasks on multiple processors using a "source transformation" approach. The assessment will be geared towards real-time systems, and therefore the hypothetical system contains aspects that will fail totally if not processed in the allocated time period.

In support of the real-time analysis of the system, a combined Target Generator and Rocket Simulator will be developed to provide inputs to, and process outputs from the Border Defense System system. Although the code in the simulator will also be written in Ada, its development will not be analyzed as part of the study.

12.1 Overview

The Border Defense System (BDS) is designed to protect a defined border against invasion. It accepts target position information from a surveillance system, and generates a real-time color graphics display for an operator to observe battlefield activity. The operator selects targets by positioning a target reticle with a pointing interface and pressing an "fire" button. These selected targets are then engaged by the BDS, which will launch a rocket and provide real-time guidance data for the rocket to the point of intercept. The graphics display is updated to indicate progress of the rocket flights (as reported by encrypted rocket messages) in real-time. Post attack assessment information is provided to the operator as well as the number of active targets and rockets.

12.2 Requirements

12.2.1 Target Display

The target display shall support a minimum of 350 x 500 picture elements (pixels) of resolution, with a minimum of 16 simultaneous colors selected from among a minimum of 256 colors. The display shall be mapped to a battle area of 4km x 4km. The BDS shall support displaying a minimum of 100 simultaneous targets. Target Report messages will be provided to the BDS from the Sensor Subsystem at a rate of 10Hz. Each report message will contain up to 100 target reports as specified in the Sensor Interface Design Document (IDD). These target reports contain target information including the target identification, classification, and position. Targets shall be displayed at the position on the display corresponding to their reported position on the battlefield. Target symbols shall be current within 250ms. Target symbols shall consist of a minimum of 8 contiguous pixels and shall be color coded to indicate target classification and engagement status. Multiple targets that map to the same screen position may appear as a single target or overlap.

12.2.2 Operator Commands

The BDS shall support input from an operator in the form of messages from a hand operated pointing device, containing a minimum of three control buttons. The current position of the pointing device shall be superimposed on the display in the form of a targeting reticle and shall be current within 50ms. The targeting reticle shall consist of the corners of a square box (minimum side of 10 pixels) with cross hairs in the center. While in "manual" mode, if the operator presses the "fire" button, the BDS shall command the Rocket Control System to launch a rocket, and will track the target closest to the pointing position for interception by that rocket. Targets that are engaged shall change color so as to appear brighter. If more than 20 rockets are active, the "fire" button will be ignored. Pressing of the "reset" button shall result in battlefield cumulative statistics being cleared. The third button shall toggle the BDS between manual and automatic modes. In automatic modes, the "fire" button shall be ignored, and target engagement shall be selected by the BDS. The target closest to the protected border shall always be selected. By default, the MANUAL mode shall be selected upon system initialization. Pressing of any single button shall result in the desired action within 250ms. When more than one button is pushed simultaneously, they shall be processed in the order of MODE, FIRE, and RESET.

12.2.3 Rocket Control

12.2.3.1 Encryption (Phase II)

All rocket data link messages shall be encrypted according to the Data Encryption Standard (DES). Encryption/Decryption shall be done on all rocket reports and rocket guidance update data within the messages. Other fields shall be transmitted as plain text. Key management shall be in accordance with "Key Control for the Border Defense System" document (Appendix J).

12.2.3.2 Rocket Reports

Rocket report lists will contain up to 20 rocket reports that have been collected by the Rocket Control System and transferred to the BDS via the Rocket Data Link. The format of the received report lists is specified in the Rocket Data Link Interface Design Document. Rocket report lists will arrive at the BDS at a 10Hz rate to provide new position data for each active (in flight) target.

12.2.3.3 Rocket Guidance

Rocket guidance lists shall be generated at a 10Hz rate and will contain new attitude control data for each rocket in flight. These lists shall be transmitted via the rocket data link to the Rocket Control System for up-link to the rockets. Each rocket will be assigned a unique (with respect to all in-flight rockets) identification number by the BDS. Upon recognition of a new rocket ID, the Rocket Control System will launch an additional rocket (up to 20 active rockets). Failure to provide rocket guidance data for each rocket within 100ms will result in the rocket performing an automatic self destruct (due to concerns about countermeasures).

12.2.4 Battle Status

Battlefield conditions and statistics shall be continuously displayed. At a minimum, the numbers of active targets and rockets, total numbers of targets destroyed and rockets expended since the last reset shall be reported on the display. The display will also clearly indicate the mode of operation (AUTOMATIC or MANUAL). Battle status shall be current within 1 second.

12.3 **Performance Requirements**

The BDS shall provide 100% hit rate while operating in the absence of effective countermeasures and with the conditions specified in the Parameter Data Base (Appendix B).

12.4 Quality Assurance Requirements

The BDS Software shall be developed in the Ada language (ANSI/MIL STD 1815A-1983) in accordance with Defense System Software Development standards (MIL-STD-2167A). No assembly language is permitted. Ada "CODE" statements may be used, but are limited to 50 code statements. A Contracting Agency approved coding style guide (Appendix F) shall be adopted and adhered to in production of all deliverable code.

13. Appendix B - Parameter Data Base

```
package PDB is
--
--| This package specifies the Parameter Data Base for the BDS
--| Simulator.  It provides a complete structure for simulator
--| parameters which can be modified on-line.
--
  grid_size  : constant := 4000;  --| 4km square

  type GRID_TYPE  is range 0..grid_size;

  type DEGREES_TYPE is digits 6 range 0.0 .. 360.0;  --| in circle

  type COUNT_TYPE is range 0..32767;         --| used for most counters

  type INTERVAL_TYPE is new DURATION range 0.001 .. 1000.0;
                               --| allowable intervals
  type RATE_TYPE is digits 5;
                               --| used for all rate calculations
------------------------------------------------------------
--   T A R G E T    I N F O R M A T I O N               --
------------------------------------------------------------
  subtype MAX_TARGETS_RANGE is COUNT_TYPE range 1..1000;
                               --|  RANGE FOR MAXIMUM NUMBER OF TARGETS

  subtype CREATE_RANGE is INTERVAL_TYPE range 0.1 .. 10.0;
                               --|  INTERVAL BETWEEN NEW TARGETS

  subtype VELOCITY_TYPE is RATE_TYPE range 0.001 .. 1000.0;
                               --| Allowable velocities in meters/sec.


  subtype VELOCITY_CHANGE_RANGE is INTERVAL_TYPE range 0.1 .. 60.0;
                          --|  INTERVAL BETWEEN CHANGES IN TARGET VELOCITY

  subtype TARGET_TURN_RATE_RANGE is RATE_TYPE range 0.01 .. 20.0;
                               --| Valid target turn rates
  type TARGET_MAX_TURN_ANGLE_RANGE_TYPE is digits 4 range 0.1 .. 360.0;
                               --| degrees
  type TARGET_KILL_RANGE_TYPE is digits 3 range 1.0 .. 100.0; --| meters
  subtype TARGET_TURN_INTERVAL_TYPE is RATE_TYPE range 1.0 .. 1000.0;
                               --| seconds between turns

  type TARGET_CLASS_TYPE is        --| Supported Target Classifications
            ( UNKNOWN,             --| Unidentified target
              T80_TANK,            --| Pact Main Battle Tank
              SA_9,                --| Mobile SAM
              BMP_2);              --| Armored Personnel Carrier
```

```
TARGET_PARAMETER_TYPE is record
  MAXIMUM_VELOCITY         : VELOCITY_TYPE;   --| travel in K-meters/hour
  AVERAGE_VELOCITY         : VELOCITY_TYPE;   --| travel in K-meters/hour
  VELOCITY_CHANGE_INTERVAL : VELOCITY_CHANGE_RANGE;  --| in seconds
  MAXIMUM_TURN_RATE        : TARGET_TURN_RATE_RANGE;  --| degrees per second
  MAX_TURN_ANGLE           : TARGET_MAX_TURN_ANGLE_RANGE; --| degrees per turn
  TURN_INTERVAL            : TARGET_TURN_INTERVAL_RANGE; --| in seconds
  VULNERABILITY_RADIUS     : TARGET_KILL_RANGE;     --| meters
end record;

MAXIMUM_TARGETS          : MAX_TARGETS_RANGE := 100;
          --| This value specifies the absolute maximum number of
          --| targets allowed to be actively reported in any one report.

TARGET_CREATE_INTERVAL   : CREATE_RANGE := create_range'last;
                  --| Indicates the rate at which targets shall be
                  --| generated to achieve the MAX_TARGETS value.

TARGET_PARAMS is array(TARGET_CLASS_TYPE) of TARGET_PARAMETER_TYPE :=
   ( UNKNOWN =>
     ( MAXIMUM_VELOCITY         =>        100.0  ,
       AVERAGE_VELOCITY         =>         30.0  ,
       VELOCITY_CHANGE_INTERVAL =>         15.0  ,
       MAXIMUM_TURN_RATE        =>         20.0  ,
       MAX_TURN_ANGLE           =>        200.0  ,
       TURN_INTERVAL            =>          2.0  ,
       VULNERABILITY_RADIUS     =>            5
     ),
     T80_TANK =>
     ( MAXIMUM_VELOCITY         =>         90.0  ,
       AVERAGE_VELOCITY         =>         50.0  ,
       VELOCITY_CHANGE_INTERVAL =>         10.0  ,
       MAXIMUM_TURN_RATE        =>         30.0  ,
       MAX_TURN_ANGLE           =>        100.0  ,
       TURN_INTERVAL            =>          5.0  ,
       VULNERABILITY_RADIUS     =>            6
     ),
     SA_9 =>   -- Gaskin Surface to Air Missile Launcher
     ( MAXIMUM_VELOCITY         =>         96.0  ,
       AVERAGE_VELOCITY         =>         67.0  ,
       VELOCITY_CHANGE_INTERVAL =>         30.0  ,
       MAXIMUM_TURN_RATE        =>         10.0  ,
       MAX_TURN_ANGLE           =>         90.0  ,
       TURN_INTERVAL            =>         15.0  ,
       VULNERABILITY_RADIUS     =>           12
     ),
     BMP_2 =>  -- Armored Troop Carrier
     ( MAXIMUM_VELOCITY         =>         59.0  ,
       AVERAGE_VELOCITY         =>         42.0  ,
       VELOCITY_CHANGE_INTERVAL =>         25.0  ,
       MAXIMUM_TURN_RATE        =>          5.0  ,
        MAX_TURN_ANGLE          =>          80.0 ,
        TURN_INTERVAL           =>          25.0 ,
        VULNERABILITY_RADIUS    =>            8
      )
    );   -- end of array

  TARGET_CREATE_RATIO is array(TARGET_CLASS_TYPE) of COUNT_TYPE :=
    (UNKNOWN => 10, T80_TANK => 50, SA_9 => 20, BMP_2 => 20);
--| numbers of each class created per 100 targets

  TARGET_REPORT_INTERVAL : INTERVAL_TYPE := 0.1;  --| seconds per report

-----------------------------------------------------------
--    R O C K E T     I N F O R M A T I O N              --
-----------------------------------------------------------


  subtype  MAX_ROCKETS_RANGE is COUNT_TYPE range 1..100;
           --| allowable range for number of rockets

  MAXIMUM_ROCKETS : MAX_ROCKETS_RANGE := 10;  --| number of active rockets

  type MASS_TYPE is digits 5 range 10.0 .. 100.0; --| rocket mass

  type THRUST_TYPE is digits 6; --| force in Newtons

  type BURN_RATE_TYPE is digits 5 range 0.001 .. 10.0; --| kg/second

  type RESISTANCE_TYPE is digits 5 range 0.001 .. 100.0; Nsec/m

  type DRIFT_VELOCITY_TYPE is digits 5 range 0.001 .. 0.5; --| meters/sec.

  subtype ROCKET_TURN_ACCEL_TYPE is new RATE_TYPE range 0.01 .. 1000.0;
           --| acceleration in degrees per second squared

  type ROCKET_PARAMETER_TYPE is record
    MASS                 : MASS_TYPE := 25.0;  --| kg (no fuel)
    FUEL                 : MASS_TYPE := 13.0;  --| kg
    THRUST               : THRUST_TYPE := 750; --| Newtons
    BURN_RATE            : BURN_RATE_TYPE := 1.0; --| kg/second
    TURN_BURN_RATE       : BURN_RATE_TYPE := 0.2; --| kg/second/degree
    FORWARD_DRAG         : RESISTANCE_TYPE := 0.4; --| Newton-seconds/meter
    SIDE_DRAG            : RESISTANCE_TYPE := 75.0; --| Newton-seconds/meter
    DRIFT                : DRIFT_VELOCITY_TYPE := 0.2; --| meters/second
    TURN_RATE            : ROCKET_TURN_ACCEL_TYPE := 300.0;--| degrees/sec
    LAUNCH_X             : GRID_TYPE := grid_size/2.0;
    LAUNCH_Y             : GRID_TYPE := 0.0;
    LAUNCH_Z             : GRID_TYPE := 0.0;
    LAUNCH_AZIMUTH       : DEGREES_TYPE := 90.0;
    LAUNCH_ELEVATION     : DEGREES_TYPE := 90.0;
  end record;
  ROCKET : ROCKET_PARAMETER_TYPE;  --| hold all rocket parameters

  ROCKET_REPORT_INTERVAL : INTERVAL_TYPE := 0.1;--| interval between reports

end PDB;  --| Parameter Data Base
```

14. Appendix C - Sensor Interface Design Document

Border Defense System (BDS)
SENSOR Interface Design Document
Release 1.0

14.1 Purpose

This document describes the interface between the BDS Sensor Subsystem (SS) and the Main Control Unit (MCU). All messages are described as well as the general protocol.

14.2 References

"The ETHERNET, A Local Area Network Data Link Layer and Physical Link Layer Specifications", Version 2.0 1982, Xerox Corporation, Stamford Connecticut.

14.3 Requirements

14.3.1 Protocol

Transmissions will be in accordance with the standard Ethernet Specifications. This applies to the Carrier Sense, Multiple Access with Collision Detection CSMA/CD mechanism as well as the binary exponential back off technique. Standard Destination and Source Address Fields, Length, Type, and Frame Check Sequence (FCS) fields shall be imposed to facilitate packet transport.

14.3.2 Error Recovery

Error recovery will be as follows: Errors shall be detected and logged. Data within an incorrect packet shall be discarded, and the previous state of the system shall be extrapolated forward in time. Multiple errors may cause performance of the system to degrade, including but not limited to loss of target representation and/or accuracy loss in rocket guidance.

14.3.3 Network Address Assignments

```
Network Address Assignments:
  Main Control Unit     = 02-60-4C-00-00-01   (hexadecimal)
  Sensor Subsystem      = 82-60-4C-00-00-02   (hexadecimal)
  Rocket DL  Subsystem  = 82-60-4C-00-00-03   (hexadecimal)
```

(Note To Facilitate Laboratory Simulation Testing, Sensor Subsystem and Rocket Data Link message addresses are "Multi-Cast" addresses and can both be received by a simulator if it is configured to receive multi cast addresses.)

14.3.4 Word Format

Byte order for data fields after the standard Ethernet header shall be transmitted Least significant byte first.

Specifically, fields after "MSG_LENGTH" shall be in a format that is directly compatible with the 80X86 family byte ordering. For example:

```
Number_of_Targets -
   Byte Offset 62: Least Significant Byte
   Byte Offset 63: Most Significant Byte
```

14.3.5 Message Summary

```
Message Name    Direction   Size (Bytes)   Rate
-------------   ---------   ------------   ------------
Target_List     SS to MCU   64..862        10Hz Nominal
Time_Sync_INIT  MCU to SS   64             0.01 Hz
-------------------------------------------------------
```

14.3.6 Message Descriptions

14.3.6.1 Target_List Message

Direction: Sensor Subsystem => Main Control Unit
Size: 64..862 Bytes
Rate: 10 Hz +/-10%
Description: The Target_List provides updated information on each target in the Area of Interest (AOI). A single time-tag is associated with all reports in the list. Up to 100 target reports may be contained in the list.

TARGET LIST MESSAGE

FIELD	SIZE	START OFFSET	RANGE	UNITS
Destination Addr	6	0	*	48-bit Net address
Source Addr	6	6	*	48-bit Net address
MSG_Type	2	12	Fixed at 5	Integer ID
MSG_Length	2	14	64..862	Integer Count
Time_Tag	4	16	0..2**32	20.11 usec
Spare	42	20	-	-
Number_of_Targets	2	62	0..100	-
The following fields are provided in accordance with the number of targets specified above.				
Target_Report01				
Target_ID01	2	64	0..32767	Integer ID
Target_X_POS01	2	66	0.0..3000.0	0.125 meter
Target_Y_POS01	2	68	0.0..5000.0	0.125 meter
Target_Class01	2	70	0..3	Target_Class
Target_Report02				
Target_ID02	2	72	0..32767	Integer ID
Target_X_POS02	2	74	0.0..3000.0	0.125 meter
Target_Y_POS02	2	76	0.0..5000.0	0.125 meter
Target_Class02	2	78	0..3	Target_Class
...	..	...	...	...
...	..	...	...	...
...	..	...	...	...
Target_Report_N				
Target_ID_N	2	8N+56	0..32767	Integer ID
Target_X_POS_N	2	8N+58	0.0..3000.0	0.125 meter
Target_Y_POS_N	2	8N+60	0.0..5000.0	0.125 meter
Target_Class_N	2	8N+62	0..3	Target_Class

* Refer to table in section 3.1

14.3.6.2 Destination Address

Refer to Ethernet Specification.

14.3.6.3 Source Address

Refer to Ethernet Specification.

14.3.6.4 Message Type

Refer to Ethernet Specification.

14.3.6.5 Message Length

Refer to Ethernet Specification.

14.3.6.6 Time Tag

This 32 bit value provides a time-of-day in 20.11 microsecond ticks. It will be roll over at midnight and reset to zero at that time. This value represents the time at which the sensor scan was taken which resulted in the following target list.

14.3.6.7 Spare

This field is reserved for future use.

14.3.6.8 Number of Targets

This 16 bit value contains the number of targets REPORTED in this message. It ranges from 0 to 100. For each target there is a Target Report.

14.3.6.9 Target Report (N)

```
A Target Report consists of:
  16 bit Target ID     - a unique number identifying a target.
  16 bit X position    - fixed point ('SMALL of 0.125) indicating
                         X offset relative to Left most Grid border.
  16 bit Y position    - fixed point ('SMALL of 0.125) indicating
                         Y offset relative to Nearest Grid border.
  16 bit Target Class  - Indicating one of four target
                         classifications as follows:

                                0 - Unknown
                                1 - T80 Battle Tank
                                2 - SA-9 "GASKIN"
                                3 - BMP-2 Infantry Combat Vehicle
```

Target reports exist for each target being tracked (up to 100). If two consecutive target reports arrive with a previously tracked target missing, it is presumed destroyed or masked. In either case, the display will no longer indicate existence of the target, and rockets on-route will be vectored to the last known position extrapolated according to their last known rate.

14.3.7 Time Synchronize/Initialize

Direction: Main Control Unit = > Sensor Subsystem
Size: 64 Bytes
Rate: 0.01 Hz
Description: The Time Synchronize/Initialize message commands the Sensor Subsystem to set its clock to the provided value and to begin (or continue) issuing target lists. All time tags received after transmission of this message shall be time-tagged with respect to this new time.

TIME SYNCHRONIZE/INITIALIZE MESSAGE

FIELD	SIZE	START OFFSET	RANGE	UNITS
Destination Addr	6	0	*	48-bit Net address
Source Addr	6	6	*	48-bit Net address
MSG_Type	2	12	Fixed at 4	Integer ID
MSG_Length	2	14	64	Integer Count
Time_Tag	4	16	0..2**32	20.11 usec

14.3.7.1 Destination Address

Refer to Ethernet Specification.

14.3.7.2 Source Address

Refer to Ethernet Specification.

14.3.7.3 Message Type

Refer to Ethernet Specification.

14.3.7.4 Message Length

Refer to Ethernet Specification.

14.3.7.5 Spare

This field is reserved for future use.

14.3.7.6 Time Tag

This 32 bit value contains the current time in 20.11 microsecond increments. Zero (0) refers to midnight Universal Time (UT).

15. Appendix D - Rocket Data Link Interface Design Document

Border Defense System (BDS)
ROCKET Data Link Interface Design Document
Release 1.0

15.1 Purpose

This document describes the interface between the BDS Rocket Data Link (RDL) and the Main Control Unit (MCU). All messages are described as well as the general protocol.

15.2 References

"The ETHERNET, A Local Area Network Data Link Layer and Physical Link Layer Specifications", Version 2.0 1982, Xerox Corporation, Stamford Connecticut.

15.3 Requirements

15.3.1 Protocol Transmissions

Protocol Transmissions will be in accordance with the standard Ethernet Specifications. This applies to the Carrier Sense, Multiple Access with Collision Detection CSMA/CD mechanism as well as the binary exponential back off technique. Standard Destination and Source Address Fields, Length, Type, and Frame Check Sequence (FCS) fields shall be imposed to facilitate packet transport.

15.3.2 Error Recovery

Error recovery will be as follows: Errors shall be detected and logged. Data within an incorrect packet shall be discarded, and the previous state of the system shall be extrapolated forward in time. Multiple errors may cause performance of the system to degrade, including but not limited to loss of target representation and/or accuracy loss in rocket guidance.

15.3.3 Network Address Assignments

```
Network Address Assignments:
    Main Control Unit    = 02-60-4C-00-00-01   (hexadecimal)
    Sensor Subsystem     = 82-60-4C-00-00-02   (hexadecimal)
    Rocket DL Subsystem  = 82-60-4C-00-00-03   (hexadecimal)
```

(Note To Facilitate Laboratory Simulation Testing, Sensor Subsystem and Rocket DL Subsystem message addresses are "Multi-Cast" addresses and can both be received by a simulator if it is configured to receive multi cast addresses.)

15.3.4 Word Format

Byte order for data fields after the standard Ethernet header shall be transmitted Least significant byte first.

Specifically, fields after "MSG_LENGTH" shall be in a format that is directly compatible with the 80X86 family byte ordering. For example:

```
Number_of_Rockets -
Byte Offset 62: Least Significant Byte
Byte Offset 63: Most Significant Byte
```

15.3.5 Message Summary

Message Name	Direction	Size (Bytes)	Rate
Rocket_Report_List	RDL to MCU	64..302	10Hz Nominal
Rocket_Guidance_List	MCU to RDL	64..862	10Hz Nominal
Time_Sync_INIT	MCU to RDL	64	0.01 Hz

15.3.6 Message Descriptions

15.3.6.1 Rocket_Report_List

Direction: Rocket Data Link => Main Control Unit
Size: 64..302 Bytes
Rate: 10 Hz +/-10%
Description: The Rocket_Report_List provides updated information on each rocket in flight. A separate time-tag is associated with each report in the list. Up to 20 rocket reports may be contained in the list. The Rocket Data Link subsystem receives messages from the rockets asynchronously over the 100ms period, and combines all of the reports into a single message for the Main Control Unit.

NOTE: The Target Report Fields in the following message are encrypted.

ROCKET REPORT LIST MESSAGE

FIELD	SIZE	START OFFSET	RANGE	UNITS
Destination Addr	6	0	*	48-bit Net address
Source Addr	6	6	*	48-bit Net address
MSG_Type	2	12	Fixed at 2	Integer ID
MSG_Length	2	14	64..862	Integer Count
Spare	46	16	-	-
Number_of_Rockets	2	62	0..20	Integer Count

\\

The following fields are provided in accordance with the number of rockets specified above.

```
\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
```

Rocket_Report01				
Time_Tag01	4	64	0..2**32	20.11 usec
Rocket_ID01	2	68	0..32767	Integer ID
Rocket_X_POS01	2	70	0.0..3000.0	0.125 meter
Rocket_Y_POS01	2	72	0.0..3000.0	0.125 meter
Rocket_Z_POS01	2	74	0.0..3000.0	0.125 meter
Rocket_Report02				
Time_Tag02	4	76	0..2**32	20.11 usec
Rocket_ID02	2	80	0..32767	Integer ID
Rocket_X_POS02	2	82	0.0..3000.0	0.125 meter
Rocket_Y_POS02	2	84	0.0..3000.0	0.125 meter
Rocket_Z_POS02	2	86	0.0..3000.0	0.125 meter
...	..	...	...	...
...	..	...	...	...
...	..	...	...	...
Rocket_Report_N				
Time_Tag_N	4	12N+52	0..2**32	20.11 usec
Rocket_ID_N	2	12N+56	0..32767	Integer ID
Rocket_X_POS_N	2	12N+58	0.0..3000.0	0.125 meter
Rocket_Y_POS_N	2	12N+60	0.0..3000.0	0.125 meter
Rocket_Z_POS_N	2	12N+62	0.0..3000.0	0.125 meter

* Refer to table in section 3.1

15.3.6.2 Destination Address

Refer to Ethernet Specification.

15.3.6.3 Source Address

Refer to Ethernet Specification.

15.3.6.4 Message Type

Refer to Ethernet Specification.

15.3.6.5 Message Length

Refer to Ethernet Specification.

15.3.6.6 Spare

This field is reserved for future use.

15.3.6.7 Number of Rockets

This 16 bit value contains the number of rockets REPORTED in this message. It ranges from 0 to 20. For each rocket there is a Rocket Report.

15.3.6.8 Rocket Report (N)

```
A Rocket Report consists of:
  32 bit Time Tag       - Time the Rocket reported its position.
  16 bit Rocket ID      - a unique number identifying a Rocket.
  16 bit X position     - fixed point ('SMALL of 0.125) indicating
                          X offset relative to Left most Grid border.
  16 bit Y position     - fixed point ('SMALL of 0.125) indicating
                          Y offset relative to Nearest Grid border.
  16 bit Z position     - fixed point ('SMALL of 0.125) indicating
                          Z offset relative to Ground Level.
```

Rocket reports exist for each rocket in flight (up to 20). If two consecutive rocket reports arrive with a previously reported rocket missing, it is presumed to have terminated flight. In this case the display will no longer indicate existence of the rocket, and no rocket guidance messages will be generated for that rocket.

15.3.7 Rocket_Guidance_List

Direction: Main Control Unit = > Rocket Data Link
Size: 64..302 Bytes
Rate: 10 Hz +/-10%
Description: The Rocket_Guidance_List provides updated direction information for each rocket in flight. The new direction is to be applied immediately by the receiving rocket. Up to 20 rocket guidance messages may be contained in the list. The Rocket Data Link subsystem will simultaneously transmit all messages to the rockets.

NOTE: The Direction Fields in the following message are encrypted.

ROCKET GUIDANCE LIST MESSAGE

FIELD	SIZE	START OFFSET	RANGE	UNITS
Destination Addr	6	0	*	48-bit Net address
Source Addr	6	6	*	48-bit Net address
MSG_Type	2	12	Fixed at 3	Integer ID
MSG_Length	2	14	64..862	Integer Count
Spare	46	16	-	-
Number_of_Rockets	2	62	0..20	Integer Count

\\

The following fields are provided in accordance with the number of rockets specified above.

Rocket_Report01				
Rocket_ID01	2	68	0..32767	Integer ID
Rocket_Azim01	2	70	-32768..32767	BAM
Rocket_Elev01	2	72	-32768..32767	BAM
Rocket_Report02				
Rocket_ID02	2	74	0..32767	Integer ID
Rocket_Azim02	2	76	-32768..32767	BAM
Rocket_Elev02	2	68	-32768..32767	BAM
...	..	...	...	...
...	..	...	...	...
...	..	...	...	...
Rocket_Report_N				
Rocket_ID_N	2	6N+62	0..32767	Integer ID
Rocket_Azim_N	2	6N+64	-32768..32767	BAM
Rocket_Elev_N	2	6N+66	-32768..32767	BAM

* Refer to table in section 3.1

15.3.7.1 Destination Address

Refer to Ethernet Specification.

15.3.7.2 Source Address

Refer to Ethernet Specification.

15.3.7.3 Message Type

Refer to Ethernet Specification.

15.3.7.4 Message Length

Refer to Ethernet Specification.

15.3.7.5 Spare

This field is reserved for future use.

15.3.7.6 Number of Rockets

This 16 bit value contains the number of rockets GUIDED in this message. It ranges from 0 to 20. For each rocket there is a Rocket Direction.

15.3.7.7 Rocket Direction (N)

A Rocket Direction consists of:

16 bit Rocket ID	- a unique number identifying a Rocket.
16 bit Azimuth	- Angle in Binary Angle Measurements with respect to the Y axis. LSB = 1/32768 degrees. Positive direction is rotation clockwise about the Z axis.
16 bit Elevation	- Angle in Binary Angle Measurements with respect to the Y axis. LSB = 1/32768 degrees. Positive direction is rotation up, about the X axis.

15.3.8 Time Synchronize/Initialize

Direction: Main Control Unit = > Rocket Data Link
Size: 64 Bytes
Rate: 0.01 Hz
Description: The Time Synchronize/Initialize message commands the Rocket Data Link to broadcast the time to all ground based rocket launchers. These units will set their clocks to the provided value. Just prior to launch, rockets will be given this time, and therefore time-tags within rocket messages will be with respect to the new time.

TIME SYNCHRONIZE/INITIALIZE MESSAGE

FIELD	SIZE	START OFFSET	RANGE	UNITS
Destination Addr	6	0	*	48-bit Net address
Source Addr	6	6	*	48-bit Net address
MSG_Type	2	12	Fixed at 0	Integer ID
MSG_Length	2	14	64	Integer Count
Time_Tag	4	16	0..2**32	20.11 usec

15.3.8.1 Destination Address

Refer to Ethernet Specification.

15.3.8.2 Source Address

Refer to Ethernet Specification.

15.3.8.3 Message Type

Refer to Ethernet Specification.

15.3.8.4 Message Length

Refer to Ethernet Specification.

15.3.8.5 Spare

This field is reserved for future use.

15.3.8.6 Time Tag

This 32 bit value contains the current time in 20.11 microsecond increments. Zero (0) refers to midnight Universal Time (UT).

16. Appendix E - Specification for the BDS Test Simulator

16.1 Scope

This document, in conjunction with the Rocket Data Link Interface Design Document (IDD), Sensor Data Link IDD, and the Parameter Data Base (PDB), specifies the requirements for the BDS Simulator system.

16.2 Applicable Documents

Reference Manual for the Ada Programming Language (ANSI/MIL-STD-1815A-1983)

Defense System Software Development (MIL-STD-2167-1985)

Rocket Data Link Interface Design Document

Sensor Data Link Interface Design Document

Parameter Data Base for the BDS Simulator

16.3 Requirements

The Simulator for the BDS system provides simulated inputs and outputs for the BDS Sensor and Rocket Interface Units. It serves to allow complete testing of the BDS in the laboratory.

16.3.1 Sensor Simulation

16.3.1.1 Target Data in Parameter Data Base

The BDS Simulator shall generate simulated sensor data in accordance with the values specified in the Parameter Data Base. These parameters shall be interpreted as follows:

16.3.1.1.1 MAXIMUM_TARGETS

This value specifies the absolute maximum number of targets allowed to be actively reported in any one report.

16.3.1.1.2 TARGET_CREATE_INTERVAL

Indicates the interval at which targets shall be generated to achieve the MAX_TARGETS value.

16.3.1.1.3 TARGET_CHARACTERISTICS

Provides the characteristics of each target class.

16.3.1.1.3.1 MAXIMUM_VELOCITY

Maximum vehicle velocity.

16.3.1.1.3.2 AVERAGE_VELOCITY

Average vehicle velocity.

16.3.1.1.3.3 VELOCITY_CHANGE_INTERVAL

Interval between target velocity changes.

16.3.1.1.3.4 MAXIMUM_TURN_RATE (angular velocity)

The maximum rate at which a vehicle can turn.

16.3.1.1.3.5 TURN_INTERVAL

The interval at which a new direction is taken.

16.3.1.1.3.6 MAX_TURN_ANGLE

Maximum turn angle for any single turn.

16.3.1.1.3.7 VULNERABILITY_RADIUS

The radius about the vehicle center that will result in destruction if a rocket strikes.

16.3.1.1.4 TARGET_REPORT_INTERVAL

Frequency of new target report messages.

16.3.1.2 Target Creation

Target creation shall occur at the TARGET_CREATE_INTERVAL until the MAXIMUM_TARGETS count has been achieved. Thereafter, targets will be created at this rate to maintain destroyed targets. Targets shall be created in the ratio of target classes specified in the PDB TARGET_CLASS_RATIO values. Target starting positions shall be initialized according to random coordinates between zero(0) and the MAXIMUM_START_X and MAXIMUM_START_Y values in the PDB. Targets are considered "active" as soon as they are created.

16.3.1.3 Target Motion

Target motion will be simulated using random numbers to provide actual motions within the maximums specified in the PDB. "Random" values shall have a normal distribution within 0.3 times the standard deviation and a scaled average of 0.5 +/-10% for samples sizes with n > 100. The random sequence period shall be greater than 2**16. Random numbers will be uniform in the range [0,1).

Target motion shall always be forward (decreasing "Y" grid offset values). Turns made by targets shall be made at the TURN_INTERVAL specified in the PDB within -0/+200ms. Vehicles will change direction at a rate that is a random factor of the MAX_TURN_RATE specified in the PDB. For the purposes of this simulation, the vehicle is assumed to achieve the maximum turn rate instantaneously. The desired turn angle shall be determined as a random factor of the MAX_TURN_ANGLE, restricted by maintaining forward motion. The change in direction shall continue until the desired turn angle is reached, or until a new turn is initiated.

Target travel shall be determined by a random factor of the AVERAGE_VELOCITY but not exceeding the MAXIMUM_VELOCITY values specified in the PDB. Target velocity shall be recomputed at the VELOCITY_CHANGE_INTERVAL specified in the PDB with -0/+200ms.

All target motion computations shall be done with sufficient accuracy so that the resultant positions/times are correct within 1 meter.

16.3.1.4 Target Reports

Target reports shall be generated at the TARGET_REPORT_INTERVAL specified in the PDB within +/- 2ms. Target reports shall be in the format specified in the Sensor Data Link IDD. Target reports shall contain an entry for each of the active targets. Targets shall be considered active from the time they are created until they are destroyed or reach a coordinate of range that is zero (Y=0).

16.3.2 Rocket Simulation

16.3.2.1 Rocket Guidance

Rocket Guidance Lists will be received from the BDS main control unit in accordance with the Rocket Data Link IDD. All rocket motions shall be simulated with respect to the time at which the Guidance Lists are received. If new rocket ID's appear in the list, the simulator shall initiate a launch for each of the new ID's. Initial rocket coordinates and vectors for launch shall be taken from the PDB parameters LAUNCH_X, LAUNCH_Y, LAUNCH_Z, LAUNCH_AZIMUTH, and LAUNCH_ELEVATION.

16.3.2.2 Rocket Report Lists

Rocket Report Lists shall be generated at the rate specified in the PDB value ROCKET_REPORT_RATE -0/+3ms. Report Lists shall contain the current simulated position of each of the active rockets. Rockets shall be considered active after receipt of a new rocket ID in a rocket guidance list until the rocket reaches a zero (0) relative altitude (Z coordinate = 0) after flight of at least 2 seconds.

16.3.2.3 Rocket Termination

Rocket Termination shall cause a simulated detonation which will destroy any simulated targets that are within their respective vulnerable radius.

16.3.2.4 Rocket Data in Parameter Data Base

The BDS Simulator shall generate simulated rocket data in accordance with the values specified in the Parameter Data Base. These parameters shall be interpreted as follows:

16.3.2.4.1 MAXIMUM_ROCKETS

Contains the maximum number of active rockets. Any new rocket ID's that arrive in guidance messages which would result in exceeding this value shall result in an error message on the operator display. Such requests for launch will be ignored.

16.3.2.4.2 ROCKET_CHARACTERISTICS

16.3.2.4.2.1 ROCKET_MASS

Take off weight of the rocket (in kg).

16.3.2.4.2.2 ROCKET_FUEL

Amount of rocket propellant (in kg).

16.3.2.4.2.3 ROCKET_THRUST

Force of rocket engine (in Newtons).

16.3.2.4.2.4 BURN_RATE

Rate at which propellant is consumed (in kg) for forward thrust.

16.3.2.4.2.5 TURN_BURN_RATE

Rate at which additional propellant is consumed for each degree of rotation/second-squared effected.

16.3.2.4.2.6 FORWARD_DRAG

Reactive force of air resistance to forward velocity.

16.3.2.4.2.7 SIDE_DRAG

Reactive force of air resistance to sideslip velocity. Note that the rocket is nearly symmetrical about its center line axis, and therefore side drag is independent of which side is traveling into the air resistance.

16.3.2.4.2.8 DRIFT

Rocket motion due to wind (m/sec). This is a maximum value to be modified by a random factor.

16.3.2.4.2.9 ROCKET_TURN_RATE

It is the maximum acceleration the rocket can achieve. For simplification, the vectored thrust for turning can be considered to have no effect on the forward thrust (that is, additional fuel is used for turning.) This fuel is burned at the rate of the TURN_BURN_RATE value.

16.3.2.4.2.10 Launch Data

LAUNCH_X, LAUNCH_Y, LAUNCH_Z, LAUNCH_AZIMUTH, and LAUNCH_ELEVATION provide the attitude and position of the rocket at launch.

16.3.2.4.3 ROCKET_REPORT_RATE

The rate at which rocket position report lists are transmitted to the BDS main control unit.

16.3.2.5 Rocket Motion

Rocket motion shall be computed at least as frequently as required by the ROCKET_REPORT_INTERVAL. Rocket motion shall compensate for drag, angular acceleration rates, gravity, changes in rocket mass and other drift due to wind. When the propellant has been expended and the rocket is still in flight, it shall continue to travel at the heading it was at when the fuel ran out.

All rocket motion computations shall be done with sufficient accuracy so that the resultant positions/times are correct within 1 meter.

16.3.3 Parameter Data Base On-Line Update

Data Base changes shall occur in a consistent fashion. Specifically, all parameters for a single rocket or sensor list operation will be static. This implies that a synchronization of the data base update with access from either the rocket simulation or sensor simulation must be performed.

Values in the PDB will be changed in accordance to the Operator Interface Requirements section of this document.

16.3.4 Scenario Generation/Playback

The BDS Simulator shall have the capability to process a previously generated target movement scenario rather than using random motions. An off-line tool shall be provided to assist in producing scenarios, and to allow selection of a particular scenario for execution. The scenario shall support a sufficient number of targets and target positions so as to efficiently utilize up to 256KB of simulator memory.

16.3.5 Time Synchronization

16.3.5.1 Sensor Subsystem

The Sensor Subsystem shall accept a Time Synchronize/Initialize command and set its internal time reference accordingly. This internal reference shall be used to supply time tags on messages and perform all time related computations.

16.3.5.2 Rocket Data Link

The Rocket Data Link shall accept a Time Synchronize/Initialize command and set its internal time reference accordingly. This internal reference shall be used to supply time tags on messages and perform all time related computations.

It is anticipated that time synchronization will be required within 100us to achieve the specified accuracies.

16.3.6 Operator Interface

16.3.6.1 Operator Display

The Operator Display shall indicate current status of the simulation as well as issue command prompts and echo operator inputs. The displayed data shall be updated at intervals not to exceed 1 second. Simulation data shall consist of a least the following items:

```
# of targets active/total
# of rockets active/total
# of destroyed targets
# of rocket misses
# of targets reaching coordinate "Y=0"

Latest Error Status
```

The format of the displayed data shall be proposed by the contractor and will require approval of the contracting agency.

16.3.6.2 Operator inputs

Operator inputs shall be menu oriented. All operator inputs shall be checked for validity, and errors will allow the operator to reenter the data. At any time, the operator shall be able to enter an <ESCAPE> (ASCII 27) character to return to the top level menu.

16.3.6.3 Parameter Data Base Variables

All Parameter Data Base variables shall be accessible to display and/or modify via the operator interface. Allowable ranges for variables shall be displayed in the data entry prompts.

16.3.7 Built In Test

Built In Test shall be performed with sufficient frequency to detect a stable hardware fault in the Encryption unit within 1 second. Such faults will be indicated on the Operator Display and invoke an automatic degraded mode of operation using software encryption. This degraded mode will reduce the number of active targets to a maximum of 10, and the number of active rockets to 1. All rockets in flight in excess of the degraded mode maximum during initiation of degraded mode will continue to be tracked. However no guidance messages will be processed and no target reports generated for these rockets.

16.4 **Quality Assurance Requirements**

16.4.1 Software Development

All software shall be developed in accordance with MIL-STD-2167. Designs shall be reviewed prior to proceeding with implementation. Code walk throughs shall be conducted on all software to insure proper design, style, and performance objectives.

16.4.2 Test Procedures

Test procedures will be written so that testing can be conducted in an automatic fashion to the greatest extent possible. Testing shall be conducted on all requirements to insure compliance with this specification and any lower level specifications (SDDD in particular).

16.4.3 Error Budgets

Error budgets for all timing and calculations shall be documented to indicate where allowable errors may occur.

16.4.4 Programming Language

The only programming language shall be Ada. No code statements shall be used without prior approval of the contracting agency. No assembly statements shall be used.

16.4.5 Implementation Dependent Features

Implementation dependent features shall be limited to the greatest extent possible. Consideration for transporting the software to a new RunTime Environment will be made when evaluating different techniques in implementation.

17. Appendix F - Ada Style Guide

Version 2.1

17.1 Introduction

This document was developed to provide guidance in the consistent generation of Ada software. If used properly, the effect should be to minimize the likelihood of coding errors, improve the readability of the code, simplify the work involved in making minor alterations to the code, and facilitate documentation development. Transportability and reusability are addressed through careful use of language features.

17.2 Identifiers

The following statements describe the standards that will apply to identifiers:

1. All Reserved word identifiers will be in lower case.

2. All type and subtype identifiers will be in upper case, and will have the suffix "_TYPE".

3. All variable identifiers will be in upper case.

4. All enumeration literals will be in upper case.

5. All access variables will have the suffix "_PTR".

6. All constant identifiers will be in lower case.

7. All other identifiers will have their first character in upper case with their remaining characters in lower case. For names containing embedded underscores, the first character after each underscore should be in upper case. Exceptions will be made for common acronyms such as "IO", they will appear in upper case: "Text_IO".

17.3 Identification, Alignment, and Spacing

The following conventions should be used:

1. Standard indentation will be two (2) spaces.

```
Example:
        if ROCKET_IN_FLIGHT then  -- Rocket is already airborne
          Enable (TRACKER);
        else
          Calibrate (SENSOR);
        end if;
```

2. Multiple conditions in an "if..." clause should be grouped together, placed on appropriate lines, and aligned so as to enhance clarity.

Example:
```
        if (OPERATOR_COMMAND = LAUNCH)  or
          (TARGET_SELECTED and AUTOMATIC_MODE) then
          Launch_Rocket;
        end if;
```

3. Each statement will be on a different line.

4. The reserved words "begin", "exception", and "end", when used in a subprogram specification, should be aligned. The code within these reserved words should be indented two spaces.

5. Loop initiators and loop terminators should be aligned. The body of the loop should be indented two spaces:

Example:
```
        loop
          RESULT := Attempt_Alignment;
          exit when RESULT = GOOD;
        end loop;
```

6. A blank space will precede and follow each operator except for exponentiation or negation.

17.4 Comments

1. A comment header will be placed at the beginning of each subprogram, package, task, or generic unit. The PDL templates should be adequate which contain:

```
--| Effects:  Describes overall effect of this code section
--| Modifies  States what global objects are modified directly
--| Requires  Indicates what initialization is required prior to invocation
--| Raises    Declares what exceptions are potentially explicitly raised and
              propagated from the subprogram
--| SDDD      Lists what Software Detailed Design Document requirements
              are fulfilled by this code section.
--| Engineer  Provides the principle designer/coder.
--| Classification (for classified projects only) Indicates security
              requirements for this code section.  Appropriate responses are:
                  TOP_SECRET, SECRET, CONFIDENTIAL, UNCLASSIFIED_SENSITIVE,
                  or UNCLASSIFIED
              The "SENSITIVE" designation applies to information that is
              ITAR Restricted, Export Controlled, Proprietary, or otherwise
              available on a "need-to-know" only basis.
```

Other standard keywords may be added to the template on a project basis.

2. An Else statement should always have an embedded comment associated with it.

```
Example:
        if ERROR then
          Cancel(LAUNCH);
        else                    --no error
          Fire (ROCKET);
        end if;
```

3. A comment line will be used to describe a relatively long set of related statements requiring special attention.

4. A blank line should proceed and follow each comment line, except for those comment lines that make up header blocks.

5. The use of embedded comments is encouraged. However, care should be taken not to obscure the code.

6. The beginning of each rendezvous statement (possibly compound) will be clearly identified with a comment consisting of a full line of dollar signs.

17.5 **PDL Compatibility**

The PDL will be expanded to produce the operational code. Therefore, there must be a way to separate the "high level" constructs from the low level code. A mechanism for this is supported by the PDL marker, which consists of "--||" at the end of an Ada statement. Ada code statements which should be included in the PDL listing should have this marker. Note that this marker should only appear in non-declarative regions.

17.6 **Restrictions**

1. The use of exception handlers to process errors from predefined exceptions that are known to be possible and can reasonably expected (i.e. hardware faults) is prohibited. They should only be used to respond to unexpected errors that are the result of software design problems or exceptions that are explicitly "raised". In this way, it may be possible to eliminate all runtime checking by using the suppress pragma if the implementation is sufficiently trustworthy.

2. "USE" clauses, in general, are allowed only for Language defined packages. (Those that appear in the Reference Manual). In cases where a large number of package-defined operators are used, then the "use" clause may be utilized ONLY for convenience of reference to operators. Other objects in packages must be referenced via the fully expanded name, or by using a "renames" statement or a subtype declaration. Objects referenced from ancestor units will use fully expanded names as well.

3. Use of Generics must be explicitly approved by the Software Manager. This is primarily to coordinate their use, rather than to discourage using generics.

4. Records with Default Discriminants are prohibited unless explicitly approved by the Software Manager. A statement indicating the performance and size impact of these structures should be included in the approval request.

5. Subprograms are limited to no more than 100 lines of code (semicolons). Compilation units should be limited to less than 500 lines of code.

6. Overloading of names (including operators) is limited to cases where the functionality is very similar, otherwise names will not be overloaded.

7. Package Specifications and Bodies will be separately compiled. Specifications will have the suffix "_S" on file names; bodies, including subunits will have the suffix "_B".

8. References to global data should be restricted to the extent possible. That is, whenever it is feasible subprograms should only operate on parameters. For efficiency reasons, global data may be accessed when necessary. This must ALWAYS be documented in the subprogram header. In cases where the same set of parameters are closely related it may be appropriate to combine them into a record type, and pass the entire record as a single parameter. Be cautioned to the fact that overhead associated with subprogram invocation can be substantial.

9. The use of tasks must be coordinated. Therefore task declarations are prohibited without explicit approval of the Software manager.

10. Use of the "GOTO" statement is prohibited unless explicitly approved by the Software Manager.

11. Units that contain subunits (i.e. parent units) should be limited to structural code that primarily invokes the subunits. This technique reduces the number of changes in the parent unit, and therefore recompilation of all of the subunits. It also keeps the parent unit at a high level relative to that of the subunits which improves comprehension of the resulting code.

12. Subprogram calls should use named parameter association in general. For example:

```
PLOT ( POSITION => TARGET_XY, COLOR => RED );
```

When many parameters are specified, they should be listed one parameter to a line, and aligned under the first parameter. For frequently used calls (or language defined subprograms) it is unnecessary to used named association. Care should be used to make the names appear meaningful and not provide redundant information.

13. Predefined numeric types should not be used directly. All numeric types should be derived from the Universal types.

14. Numeric literals other than 0 or 1, shall not appear outside of declarative regions. (That is, be embedded in the executable subprogram bodies.) Even these literals will only appear if they are used to initialize or increment a counter.

17.7 Naming Conventions

Names should be selected to be meaningful to anyone trying to read the code, not just the original designer. This implies that it is inappropriate to use abbreviations that are not clearly understood. Names should be kept under 20 characters in length when possible, in recognition of the fact that very large names can detract from comprehension when it is difficult to understand the structure because of clutter caused by huge names. For example:

Item	Good	Poor
List of targets	TARGET_LIST	TGTLST
Message Buffer	MSG_BUFFER	MSGBF
Mouse (user interface) Report	MOUSE_REPORT	RAT_MSG
Ethernet Communications Driver	NET_DRIVER	ETHERNET_COMMUNICATIONS_DRIVER

A list of appropriate "standard" acronyms and names should be maintained for each project. Abbreviations such as MSG for message are acceptable provided they are in (or added to) the project list. Consistency should be maintained among all of the developed software for a project.

18. Appendix G - Sample Timing Budget

Border Defense System - Budgeted Times for Major Processing

Function Name	Average Iterations per second	Time per Iteration (microsec)	Required Time (microsec)
Receive Rocket Report List	10	200	2000
Decode Rocket Report	200	800	160000
Compute Rocket Attitude	200	2000	400000
Encode Rocket Update	200	800	160000
Transmit Rocket Update List	10	200	2000
Display Rocket Position	200	1200	240000
Receive Target List	10	200	2000
Display Target Position	1000	725	1068000
Receive Mouse Character	175	27	4725
Update Cursor Position	35	2000	70000
Process Launch Request	2	500	1000
Update Statistics Display	1	1000	1000
TOTAL			1767725

Notes:

1) Total time indicates greater than 100% utilization for a single processor.

2) # of interrupts/second = 205

3) Total activities/second = 2040

The following listing is an example of a time stamp trace for the Rocket.Control task. It indicates that the task operates on an accurate 100ms periodic basis until enough rockets are airborne so that it begins missing deadlines (marked with an *).

```
--$TP(0002) Control task start time  229934 Delta =   229934 microseconds
--$TP(0002) Control task start time  330111 Delta =   100177 microseconds
--$TP(0002) Control task start time  429700 Delta =   99589 microseconds
--$TP(0002) Control task start time  530269 Delta =   100569 microseconds
--$TP(0002) Control task start time  629862 Delta =   99593 microseconds
--$TP(0002) Control task start time  729451 Delta =   99589 microseconds
--$TP(0002) Control task start time  830020 Delta =   100569 microseconds
--$TP(0002) Control task start time  929610 Delta =   99589 microseconds
--$TP(0002) Control task start time  1029201 Delta =   99592 microseconds
--$TP(0002) Control task start time  1129767 Delta =   100566 microseconds
--$TP(0002) Control task start time  1229360 Delta =   99593 microseconds
--$TP(0002) Control task start time  1329925 Delta =   100566 microseconds
--$TP(0002) Control task start time  1429517 Delta =   99592 microseconds
--$TP(0002) Control task start time  1529107 Delta =   99590 microseconds
--$TP(0002) Control task start time  1629676 Delta =   100568 microseconds
--$TP(0002) Control task start time  1729265 Delta =   99589 microseconds
--$TP(0002) Control task start time  1828859 Delta =   99594 microseconds
--$TP(0002) Control task start time  1929425 Delta =   100567 microseconds
--$TP(0002) Control task start time  2029015 Delta =   99590 microseconds
--$TP(0002) Control task start time  2129585 Delta =   100570 microseconds
--$TP(0002) Control task start time  2229175 Delta =   99589 microseconds
--$TP(0002) Control task start time  2328766 Delta =   99592 microseconds
--$TP(0002) Control task start time  2429333 Delta =   100567 microseconds
--$TP(0002) Control task start time  2528926 Delta =   99593 microseconds
--$TP(0002) Control task start time  2628516 Delta =   99590 microseconds
--$TP(0002) Control task start time  2729083 Delta =   100567 microseconds
--$TP(0002) Control task start time  2828673 Delta =   99590 microseconds
--$TP(0002) Control task start time  2929242 Delta =   100569 microseconds
--$TP(0002) Control task start time  3028832 Delta =   99589 microseconds
--$TP(0002) Control task start time  3128422 Delta =   99590 microseconds
--$TP(0002) Control task start time  3228989 Delta =   100567 microseconds
--$TP(0002) Control task start time  3328579 Delta =   99590 microseconds
--$TP(0002) Control task start time  3428173 Delta =   99594 microseconds
--$TP(0002) Control task start time  3528740 Delta =   100567 microseconds
--$TP(0002) Control task start time  3628331 Delta =   99591 microseconds
--$TP(0002) Control task start time  3728900 Delta =   100569 microseconds
--$TP(0002) Control task start time  3828491 Delta =   99591 microseconds
--$TP(0002) Control task start time  3928081 Delta =   99590 microseconds
--$TP(0002) Control task start time  4028648 Delta =   100567 microseconds
--$TP(0002) Control task start time  4128239 Delta =   99591 microseconds
--$TP(0002) Control task start time  4227829 Delta =   99590 microseconds
--$TP(0002) Control task start time  4329371 Delta =   101542 microseconds
--$TP(0002) Control task start time  4427989 Delta =   98617 microseconds
--$TP(0002) Control task start time  4528555 Delta =   100567 microseconds
--$TP(0002) Control task start time  4628146 Delta =   99591 microseconds
--$TP(0002) Control task start time  4728713 Delta =   100567 microseconds
```

```
--$TP(0002) Control task start time  4828306 Delta =   99594 microseconds
--$TP(0002) Control task start time  4927896 Delta =   99590 microseconds
--$TP(0002) Control task start time  5027487 Delta =   99590 microseconds
--$TP(0002) Control task start time  5128054 Delta =   100567 microseconds
--$TP(0002) Control task start time  5227645 Delta =   99591 microseconds
--$TP(0002) Control task start time  5328273 Delta =   100628 microseconds
--$TP(0002) Control task start time  5427802 Delta =   99530 microseconds
--$TP(0002) Control task start time  5527394 Delta =   99591 microseconds
--$TP(0002) Control task start time  5634545 Delta =   107152 microseconds   <<==  *
--$TP(0002) Control task start time  5727554 Delta =   93008 microseconds
--$TP(0002) Control task start time  5827144 Delta =   99590 microseconds
--$TP(0002) Control task start time  5933623 Delta =   106479 microseconds
--$TP(0002) Control task start time  6027303 Delta =   93680 microseconds
--$TP(0002) Control task start time  6127870 Delta =   100567 microseconds
--$TP(0002) Control task start time  6233782 Delta =   105912 microseconds
--$TP(0002) Control task start time  6330899 Delta =   97117 microseconds
--$TP(0002) Control task start time  6432494 Delta =   101595 microseconds
--$TP(0002) Control task start time  6533338 Delta =   100844 microseconds
--$TP(0002) Control task start time  6638533 Delta =   105195 microseconds
--$TP(0002) Control task start time  6740030 Delta =   101498 microseconds
--$TP(0002) Control task start time  6841556 Delta =   101526 microseconds
--$TP(0002) Control task start time  7029347 Delta =   187791 microseconds
--$TP(0002) Control task start time  7132627 Delta =   103279 microseconds
--$TP(0002) Control task start time  7242113 Delta =   109486 microseconds
--$TP(0002) Control task start time  7417546 Delta =   175433 microseconds
--$TP(0002) Control task start time  7529064 Delta =   111518 microseconds
--$TP(0002) Control task start time  7635551 Delta =   106488 microseconds
--$TP(0002) Control task start time  7742059 Delta =   106508 microseconds
--$TP(0002) Control task start time  7919917 Delta =   177857 microseconds
--$TP(0002) Control task start time  8027722 Delta =   107805 microseconds
--$TP(0002) Control task start time  8141697 Delta =   113975 microseconds
--$TP(0002) Control task start time  8307434 Delta =   165737 microseconds
--$TP(0002) Control task start time  8423121 Delta =   115687 microseconds
--$TP(0002) Control task start time  8534182 Delta =   111061 microseconds
--$TP(0002) Control task start time  8697240 Delta =   163058 microseconds
--$TP(0002) Control task start time  8815014 Delta =   117774 microseconds
--$TP(0002) Control task start time  8928160 Delta =   113146 microseconds
--$TP(0002) Control task start time  9040253 Delta =   112092 microseconds
--$TP(0002) Control task start time  9205898 Delta =   165646 microseconds
--$TP(0002) Control task start time  9321093 Delta =   115195 microseconds
--$TP(0002) Control task start time  9436134 Delta =   115041 microseconds
--$TP(0002) Control task start time  9592926 Delta =   156792 microseconds
--$TP(0002) Control task start time  9707996 Delta =   115070 microseconds
--$TP(0002) Control task start time  9823226 Delta =   115230 microseconds
--$TP(0002) Control task start time  9938537 Delta =   115311 microseconds
--$TP(0002) Control task start time  10096715 Delta =   158178 microseconds
--$TP(0002) Control task start time  10211935 Delta =   115220 microseconds
--$TP(0002) Control task start time  10327281 Delta =   115346 microseconds
```

19 Appendix H - Border Defense System Ada Source Code

The source code for the BDS system follows in alphabetical order of the unit names (specifications precede bodies).

```
-------------------------------------------------------------------------
--| UNIT:     BDS Spec & Body.                                          --
--| Effects:  Initiates main processing, loops recording idle time.    --
--| Modifies: No global data is modified.                              --
--| Requires: Status.Initialize be called before Mouse.Initialize.     --
--| Raises:   No explicitly raised exceptions are propagated.          --
--| Engineer: T. Griest                                                --
-------------------------------------------------------------------------
--**************** Distribution and Copyright ***************************
-- Derivation    : LabTek Border Defense System V1.0                    *
--                                                                       *
-- This Border Defense System Software inherits the LabTek copyright.  *
-- The following copyright must be included in all software utilizing  *
-- this application program.                                            *
--                                                                       *
-- Copyright 1989 by LabTek Corporation, Woodbridge, CT, USA            *
--                                                                       *
-- Permission to use, copy, modify, and distribute this                 *
-- software and its documentation for any purpose and without           *
-- fee is hereby granted, provided that the above copyright             *
-- notice appear in all copies and that both that copyright             *
-- notice and this permission notice appear in supporting               *
-- documentation, and that the name of LabTek not be used in            *
-- advertising or publicity pertaining to distribution of the           *
-- software without specific, written prior permission.                 *
-- LabTek makes no representations about the suitability of             *
-- this software for any purpose.  It is provided "as is"               *
-- without express or implied warranty.                                 *
--                                                                       *
--**************** Disclaimer ********************************************
--                                                                       *
-- This software and its documentation are provided "AS IS" and         *
-- without any expressed or implied warranties whatsoever.              *
-- No warranties as to performance, merchantability, or fitness         *
-- for a particular purpose exist.                                      *
--                                                                       *
-- In no event shall any person or organization of people be            *
-- held responsible for any direct, indirect, consequential             *
-- or inconsequential damages or lost profits.                          *
--                                                                       *
--**************** END-PROLOGUE ******************************************


with Config;            -- global configuration parameters
with Status;            -- updates statistics used
with Types;             -- global types definitions
with Mouse;             -- mouse movement and rocket launching
with Rocket;            -- rocket attitude and aimpoint calculations
with Target;            -- generation of various targets
with Interrupt_Control; -- enabling and disabling of (all) interrupts
with Machine_Dependent; -- individual pixel plotting for EGA
with Time_Stamp;         -- run time profiler
        pragma ELABORATE(Mouse, Status, Interrupt_Control, Time_Stamp);

procedure BDS is
-- This is the main program for the Border Defense System.  It has only
-- two calls which are of any importance, i.e., the other code is for
-- timing purposes only.  The first call performs initialization of the screen
-- statistics descriptions and their initial values.  The second call starts
-- the mouse.

use Types;                                  -- for visibility to "+"

pragma PRIORITY(Config.bds_priority);

COUNT : Types.WORD;                         -- these two variables are for
SLOW  : Types.WORD;                         -- slowing the time stamps

begin
  Status.Initialize;                        -- print screen statistics
  Mouse.Initialize;                         -- must be done after status signal
  loop                                      -- done with initialization
    --$TP(0001) BDS main time stamp
    SLOW := 1;
    for COUNT in 1..2000 loop
      SLOW := SLOW + 1;
    end loop;
  end loop;
end BDS;
```

```
-------------------------------------------------------------------------
--| UNIT:     Config Spec.                                              --
--| Effects:  Provides system-wide configuration constants.             --
--| Modifies: No global data is modified.                               --
--| Requires: No initialization is required.                            --
--| Raises:   No explicitly raised exceptions are propagated.           --
--| Engineer: T. Griest.                                                --
-------------------------------------------------------------------------


-- Date    : 10-11-88
-- Purpose :
--   This package contains global configuration parameters.  It is referenced
-- for ALL parameters which are likely to be changed during system maintenance.

with System;

package Config is

-- The following two constants allow the space needed for the various tasks to
-- be declared in bytes.
byte                      : constant := 8;        -- 8 bits
bytes_per_storage_unit    : constant := byte / System.STORAGE_UNIT;

-- Now define battlefield area perimeters

meters_in_battle_area     : constant := 4_000.0; -- in X and Y direction

meters_per_X_pixel        : constant :=  9.625;  -- rounded up to nearest
meters_per_Y_pixel        : constant := 11.875;  -- Types.METER.
max_pixels_in_battle_area : constant := meters_in_battle_area
                                        / meters_per_X_pixel;

max_active_symbols        : constant := 200;     -- max simultaneous symbols

--
-- Task priorities in order of decreasing urgency.
--
--  NOTE: MOUSE IN_CHAR has no priority because it runs
--  completely at the hardware interrupt level.
--  The idea implemented here is that all the Simulator information is
--  of higher priority than the actual Border Defense System code.
save_priority             : constant := 20;      -- Mouse_Buffer
display_priority          : constant := 18;      -- Graphics
track_data_priority       : constant := 16;      -- Target
report_buf_priority       : constant :=  9;      -- Simulator
guide_buf_priority        : constant :=  9;      -- Simulator
rock_sup_priority         : constant :=  8;      -- Simulator
targ_sup_priority         : constant :=  7;      -- Simulator
control_priority          : constant :=  5;      -- Rocket
guidance_priority         : constant :=  4;      -- Rocket
```

```
track_priority          : constant :=  3;       -- Target
update_priority         : constant :=  2;       -- Status
hds_priority            : constant :=  1;       -- Main

--
-- define entire hi-res screen display borders.  The screen is divided into
-- two main sections.  There is the battlefield area where the targets, rockets,
-- and reticle are allowed to move, and there is the statistics area where our
-- current statistics will be displayed.  The maximum number of digits allowed
-- in any statistics displayed is statistics_length.  Between the statistics and
-- the battlefield there is a border.
--
--
-- define entire screen constants
--
entire_screen_left      : constant := 0;
entire_screen_right     : constant := 639;
entire_screen_top       : constant := 0;
entire_screen_bottom    : constant := 349;
--
-- define battlefield display borders and center.
--
battlefield_screen_left   : constant := 222;    -- starting (left)
battlefield_screen_right  : constant := 638;    -- ending (in pixels)
battlefield_screen_top    : constant := 1;      -- starting (top)
battlefield_screen_bottom : constant := 338;    -- ending (in pixels)
battlefield_center_x      : constant := 430;    --
battlefield_center_y      : constant := 169;    --
--
-- define border between battlefield and statistics.
--
border_left             : constant := 221;      -- starting (left)
border_right            : constant := 639;      -- ending (in pixels)
border_top              : constant := 0;        -- starting (top)
border_bottom           : constant := 339;      -- ending (in pixels)
--
-- define statistics display borders.
--
status_left             : constant := 0;        -- starting (left)
status_right            : constant := 220;      -- ending (in pixels)
status_top              : constant := 0;        -- starting (top)
status_bottom           : constant := 349;      -- ending (in pixels)
--
-- statistics_length is the number of digits allowed in any status field, and
-- stats_title_max_length is the max number of letters any particular
-- statistics title may contain.
--
statistics_length       : constant := 4;
stats_title_max_length  : constant := 11;

max_targets             : constant := 100;      -- total targets
```

```
max_rockets          : constant := 20;      -- total rockets

interval             : constant := 0.100;   -- basic interval is 100ms

--  launch attitude
launch_azimuth       : constant := 16384;   -- straight ahead in BAMS
launch_elevation     : constant := 16384;   -- straight up  in BAMS

kill_radius          : constant := 10.0;    -- 10 meters x 10 meters

end Config;
```

```
----------------------------------------------------------------------
--| UNIT:     Control task subunit.                                  --
--| Effects:  Provides overall control for rocket flight and display. --
--| Modifies: Updates rocket data base in Rocket body.               --
--| Requires: No initialization is required.                         --
--| Raises:   No explicitly raised exceptions are propagated.        --
--| Engineer: T. Griest.                                             --
----------------------------------------------------------------------


with Interrupt_Control;
with Grid_to_Pixel;
with Simulate;
with Target;
with Calendar;
with Engage;
with Time_Stamp;
 pragma ELABORATE(Interrupt_Control, Grid_to_Pixel,
                  Simulate, Target, Calendar, Engage, Time_Stamp);

separate(Rocket)
--
--  The Control task accepts data from the Rocket Data Link,
--  provides information to the Display task, and disperses information
--  to each of the guidance tasks.  It then collects the computations
    of the guidance tasks, and generates a guidance message to be sent
--  back to the Rocket Data Link
--

task body Control_Type is

  use Calendar;           -- for operators
  use Types;              -- for operators

  package RDL renames Simulate.RDL;   -- make simulator transparent

  dis_list_size    : constant := Config.max_rockets;

  MOVE_NUMBER      : Types.WORD_INDEX;
                                          --| to update display

  NEXT_ROCKET_MSG  : ROCKET_MSG_TYPE;          --| local copy of input msg
  NEXT_TARGET_LIST : Target.TARGET_DATA_LIST_TYPE; --| local copy of input data
  GUIDE_MSG        : ROCKET_GUIDE_MSG_TYPE;    --| local copy of output msg

  AIMPOINT_LIST    : AIMPOINT_LIST_TYPE(Types.ROCKET_INDEX_TYPE);
                                          --| local copy
  MOVE_ROCKETS     : Graphics.MOVE_LIST_TYPE(Types.ROCKET_INDEX_TYPE);
  MOVE_INDEX       : Types.WORD_INDEX;

  PIXEL_POINT      : Shapes.PIXEL;
```

```
MSG_INDEX         : Types.WORD_INDEX;          --| used to index incoming report
OLD_TIME_TAG      : Types.WORD;                --| to filter stale reports out
ANY_ACTIVE_ROCKETS : BOOLEAN := FALSE;         --| used to update OLD_TIME_TAG
ACTIVE_ROCKETS_ID  : Types.ROCKET_INDEX_TYPE; --| holds an active rockets ID

NEXT_ENGAGED      : Target.TARGET_ID_TYPE;
NEXT_DISENGAGED   : Target.TARGET_ID_TYPE;     --| keep track of all disengagements

DISENGAGED_LIST   : array(Types.ROCKET_INDEX_TYPE) of Target.TARGET_ID_TYPE;
DISENGAGED_ON_PTR  : Types.WORD_INDEX;
DISENGAGED_OFF_PTR : Types.WORD_INDEX;
DISENGAGED_ACK_PTR : Types.WORD_INDEX;

AVAILABLE_ROCKET : Types.WORD_INDEX;           --| possible rocket to launch

LAUNCH_PENDING    : BOOLEAN := FALSE;
LAUNCH_TARGET     : Target.TARGET_ID_TYPE;
LAUNCH_ROCKET     : Types.ROCKET_INDEX_TYPE;

ROCKET_DESTROYED : BOOLEAN;
ROCKET_LAUNCHED  : BOOLEAN;

START_TIME        : Calendar.TIME;
DELAY_PERIOD      : DURATION;

begin
  for I in ROCKET_HISTORY'range loop      --| initialize track data
    ROCKET_HISTORY(I).ACTIVE := FALSE;
    DISENGAGED_LIST(I) := 0;
  end loop;
  NEXT_ENGAGED := 0;

  DISENGAGED_ON_PTR  := 1;                --| initialize disengage circle queue
  DISENGAGED_OFF_PTR := 1;
  DISENGAGED_ACK_PTR := 1;

  OLD_TIME_TAG := 0;

  START_TIME := Calendar.CLOCK;

  loop                                    --| Main processing loop
    begin                                   --| exception block
      START_TIME := START_TIME + Config.interval;
      --$TP(0002) Control task start time
      ROCKET_DESTROYED := FALSE;
      ROCKET_LAUNCHED  := FALSE;
      ANY_ACTIVE_ROCKETS := FALSE;
--
-- Rendezvous with buffer task to get next rocket message from sensor
--
      --$TP(0003) Control rendezvous with Report_Buf start
      RDL.Report_Buf.Get_Report(NEXT_ROCKET_MSG);
      --$TP(0004) Control rendezvous with Report_Buf end
--
--  Rendezvous to Get target list from target tracker, and provide it
--  with information on which targets have been engaged and disengaged.
--
--  If there are more on circular disengage queue, send another to tracker
--
      if DISENGAGED_OFF_PTR /= DISENGAGED_ON_PTR then
        NEXT_DISENGAGED := DISENGAGED_LIST(DISENGAGED_OFF_PTR);
        DISENGAGED_OFF_PTR := DISENGAGED_OFF_PTR rem dis_list_size + 1;
      else
        NEXT_DISENGAGED := 0;
      end if;

      --$TP(0005) Control rendezvous with Track_Dat start
--$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
      Target.Track_Data.Get(NEXT_TARGET_LIST, NEXT_ENGAGED, NEXT_DISENGAGED);
      --$TP(0006) Control rendezvous with Track_Dat end
--
--  Check if Track task has recognized the engage request, if so then
--  it is safe to clear it, and possibly engage another.
--
      if NEXT_ENGAGED /= 0 and then
         NEXT_TARGET_LIST(NEXT_ENGAGED).STATUS.ENGAGED
      then
        NEXT_ENGAGED := 0;
      end if;
--
--  Check to see if last disengage request was acknowledged
--
      if DISENGAGED_ACK_PTR /= DISENGAGED_OFF_PTR and then
        not NEXT_TARGET_LIST(DISENGAGED_LIST(DISENGAGED_ACK_PTR)).STATUS.ENGAGED
      then
        DISENGAGED_ACK_PTR := DISENGAGED_ACK_PTR rem dis_list_size + 1;
      end if;


--
--| determine which rockets have been expended, and delete them from screen
--| (previously active, but no longer in report list)
--
      MOVE_INDEX := 0;
      MSG_INDEX := 1;
      for ROCKET_ID in Types.ROCKET_INDEX_TYPE loop

        if ROCKET_HISTORY(ROCKET_ID).ACTIVE then
          if NEXT_ROCKET_MSG.ROCKET_LIST(MSG_INDEX).TIME_TAG = OLD_TIME_TAG then
            ANY_ACTIVE_ROCKETS := TRUE;
            ACTIVE_ROCKETS_ID := ROCKET_ID;
            exit;                                   -- old rocket report
          end if;
```

```
--
--  look at most recent rocket report message to make sure rocket is still alive
--

          if MSG_INDEX <= NEXT_ROCKET_MSG.NUM_ROCKETS and then
             ROCKET_ID = NEXT_ROCKET_MSG.ROCKET_LIST(MSG_INDEX).ROCKET_ID
          then
            ROCKET_HISTORY(ROCKET_ID).POSITION_PAIR.ROCKET_OLD :=
                              ROCKET_HISTORY(ROCKET_ID).POSITION_PAIR.ROCKET_NEW;
            ROCKET_HISTORY(ROCKET_ID).POSITION_PAIR.ROCKET_NEW :=
                               NEXT_ROCKET_MSG.ROCKET_LIST(MSG_INDEX).POSITION;
            ROCKET_HISTORY(ROCKET_ID).POSITION_PAIR.TARGET_OLD :=
                              ROCKET_HISTORY(ROCKET_ID).POSITION_PAIR.TARGET_NEW;
            ROCKET_HISTORY(ROCKET_ID).POSITION_PAIR.TARGET_NEW :=
                NEXT_TARGET_LIST(ROCKET_HISTORY(ROCKET_ID).TARGET).POSITION_NEW;
            MOVE_INDEX := MOVE_INDEX + 1;
            MOVE_ROCKETS(MOVE_INDEX) :=
(XY_OLD => Grid_to_Pixel(ROCKET_HISTORY(ROCKET_ID).POSITION_PAIR.ROCKET_OLD),
 XY_NEW => Grid_to_Pixel(ROCKET_HISTORY(ROCKET_ID).POSITION_PAIR.ROCKET_NEW),
 OBJECT => (Shapes.PIXEL_MODE,Shapes.ROCKET),
 COLOR  => Graphics.ROCKET_COLOR);
            MSG_INDEX := MSG_INDEX + 1;
          else
--
--  the rocket has deceased, put it in the list for erasure.
--
            PIXEL_POINT := Grid_to_Pixel(     -- get last point in pixel value
                          ROCKET_HISTORY(ROCKET_ID).POSITION_PAIR.ROCKET_NEW);
            ROCKET_HISTORY(ROCKET_ID).ACTIVE := FALSE; -- mark as inactive
            MOVE_INDEX := MOVE_INDEX + 1;
            MOVE_ROCKETS(MOVE_INDEX) :=
              (PIXEL_POINT,
               PIXEL_POINT,
               (Shapes.PIXEL_MODE,Shapes.ROCKET),
               Graphics.background_color);
            AVAILABLE_ROCKET := ROCKET_ID;        -- save if decide to launch
            DISENGAGED_LIST(DISENGAGED_ON_PTR):= ROCKET_HISTORY(ROCKET_ID).TARGET;
            DISENGAGED_ON_PTR  := DISENGAGED_ON_PTR rem dis_list_size + 1;
            Interrupt_Control.Disable;
            Status.STATUS_CONTROL(Status.AIRBORNE).DATA :=
               Status.STATUS_CONTROL(Status.AIRBORNE).DATA - 1;
            Status.STATUS_CONTROL(Status.EXPENDED).DATA :=
               Status.STATUS_CONTROL(Status.EXPENDED).DATA + 1;
            Interrupt_Control.Enable;
            ROCKET_DESTROYED := TRUE;
          end if;   -- found
        else
--
--    rocket slot previously inactive, see if rocket has launched
--
```

```
        if MSG_INDEX <= NEXT_ROCKET_MSG.NUM_ROCKETS and then
           NEXT_ROCKET_MSG.ROCKET_LIST(MSG_INDEX).ROCKET_ID =
                                                   ROCKET_ID
        then
--
--    ROCKET HAS BEEN LAUNCHED, UPDATE DATA BASE
--
          ROCKET_HISTORY(ROCKET_ID) :=
             ( TRUE,                        -- ACTIVE
               LAUNCH_TARGET,               -- TARGET
               ( NEXT_ROCKET_MSG.ROCKET_LIST(MSG_INDEX).POSITION, -- NEW
                 NEXT_ROCKET_MSG.ROCKET_LIST(MSG_INDEX).POSITION, -- OLD
                 NEXT_TARGET_LIST(LAUNCH_TARGET).POSITION_NEW,    -- NEW
                 NEXT_TARGET_LIST(LAUNCH_TARGET).POSITION_NEW)    -- OLD
             );
          LAUNCH_PENDING := FALSE;               -- all accounted for
          MSG_INDEX := MSG_INDEX + 1;
          Interrupt_Control.Disable;
          Status.STATUS_CONTROL(Status.AIRBORNE).DATA :=
             Status.STATUS_CONTROL(Status.AIRBORNE).DATA + 1;
          Interrupt_Control.Enable;
          ROCKET_LAUNCHED := TRUE;
        else
          AVAILABLE_ROCKET := ROCKET_ID;
        end if;  -- new rocket test
      end if;    -- active test
    end loop;    -- rocket-id loop (scan of all rockets)
--
--| Update Time tag for next message.
--
    if ANY_ACTIVE_ROCKETS then
      OLD_TIME_TAG := NEXT_ROCKET_MSG.ROCKET_LIST(ACTIVE_ROCKETS_ID).TIME_TAG;
    end if;                 -- if no active rockets, don't change OLD_TIME_TAG.


--
--| Get guidance task(s) working on finding new aimpoint for guidance msg
--
    for I in Types.WORD_INDEX range 1..Distrib.num_guide_tasks loop
      --$TP(0007) Control rendezvous with Guidance(1) start
      Rocket_Guide(I).History(
         ROCKET_HISTORY(Distrib.guide_low(I)..Distrib.guide_high(I)));
      --$TP(0008) Control rendezvous with Guidance(1) end
    end loop;
--
--| update status information
--
    Interrupt_Control.Disable;
    if ROCKET_LAUNCHED then
      Status.STATUS_CONTROL(Status.AIRBORNE).DISPLAYED := FALSE;
    end if;
    if ROCKET_LAUNCHED or ROCKET_DESTROYED then
```

```
        Status.STATUS_CONTROL(Status.AIRBORNE).DISPLAYED := FALSE;
        Status.STATUS_CONTROL(Status.EXPENDED).DISPLAYED := FALSE;
        Status.REQ_COUNT := Status.REQ_COUNT + 1;
        if Status.REQ_COUNT = 1 then
          --$TP(0009) Control rendezvous with Status start
          Status.Update.Signal;
          --$TP(0010) Control rendezvous with Status end
        end if;
      end if;
      Interrupt_Control.Enable;

      MSG_INDEX := 0;              -- zero index for creating guidance message
--
--  Now, check if we should try to create a new ROCKET.  Note that
--  if a rocket has just been destroyed, don't try to fire a new one
--  before the rocket tracker knows that it has been disengaged.  Otherwise
--  it is likely to choose a target other than one that is closest.
--
      if not LAUNCH_PENDING and
         DISENGAGED_ACK_PTR = DISENGAGED_ON_PTR and    -- all have been ack'ed
         NEXT_ENGAGED = 0                              -- engage has been ack'ed
      then
        NEXT_ENGAGED := Engage(NEXT_TARGET_LIST);
        if NEXT_ENGAGED > 0 then
          LAUNCH_ROCKET := AVAILABLE_ROCKET;
          LAUNCH_TARGET := NEXT_ENGAGED;
          LAUNCH_PENDING := TRUE;
        end if;  -- ready to launch
      end if;   -- not pending check
--
--| get graphics task working on displaying rockets
--
      --$TP(0011) Control rendezvous with Graphics start
      Graphics.Display.Move(Graphics.LOW, MOVE_ROCKETS(1..MOVE_INDEX));
      --$TP(0012) Control rendezvous with Graphics end
--
--| now get results of guidance information
--
      for I in Types.WORD_INDEX range 1..Distrib.num_guide_tasks loop
        --$TP(0013) Control rendezvous with Guidance(2) start
        Rocket_Guide(I).Next_Guidance(
          AIMPOINT_LIST(Distrib.guide_low(I)..Distrib.guide_high(I)) );
        --$TP(0014) Control rendezvous with Guidance(2) end
      end loop;

--
--  Now generate new guidance message and send to Guide_Buf
--
--
      for ROCKET_ID in ROCKET_HISTORY'RANGE loop
        if ROCKET_HISTORY(ROCKET_ID).ACTIVE then
```

```
          MSG_INDEX := MSG_INDEX + 1;
          GUIDE_MSG.ROCKET_GUIDE_LIST(MSG_INDEX) :=
                                       (ROCKET_ID,AIMPOINT_LIST(ROCKET_ID));
        elsif LAUNCH_PENDING and then
              ROCKET_ID = LAUNCH_ROCKET then
          MSG_INDEX := MSG_INDEX + 1;
-- initiate launch
          GUIDE_MSG.ROCKET_GUIDE_LIST(MSG_INDEX) := (ROCKET_ID,
                                    (Config.launch_azimuth,
                                     Config.launch_elevation));
        end if;
      end loop;
      GUIDE_MSG.NUM_ROCKETS := MSG_INDEX;
      --$TP(0015) Control rendezvous with Guide_Buf start
      RDL.Guide_Buf.Put_Guide(GUIDE_MSG);   -- send new guidance message
      --$TP(0016) Control rendezvous with Guide_Buf end
--
--  periodic schedule
--
      DELAY_PERIOD := START_TIME - Calendar.Clock;
      if DELAY_PERIOD < 0.0 then
        START_TIME := Calendar.Clock;
      end if;
      --$TP(0017) Control task end time

      delay DELAY_PERIOD;

    exception
      when others =>
        Debug_IO.Put_Line("Exception in Control task");
    end;                                -- exception block
  end loop;                             --| main processing loop
end Control_Type;   -- Rocket.Control task body
```

```
------------------------------------------------------------------
--| UNIT:     Distrib Package Spec.                             --
--| Effects:  Provides parameters to control task arrays and work lists.--
--| Modifies: No global data is modified.                       --
--| Requires: No initialization is required.                    --
--| Raises:   No explicitly raised exceptions are propagated.   --
--| Engineer: T. Griest.                                        --
------------------------------------------------------------------


with types;
--------------------------------------------------
--  DISTRIBUTION CONTROL PARAMETERS             --
--------------------------------------------------
package Distrib is
  num_guide_tasks  : constant := 1;       -- for now
  guide_low        : constant array(Types.WORD_INDEX range 1..num_guide_tasks)
                       of Types.WORD_INDEX := (others=>1);
  guide_high       : constant array(Types.WORD_INDEX range 1..num_guide_tasks)
                       of Types.WORD_INDEX := (others=>20);
end Distrib;
```

```
---------------------------------------------------------------------
--| UNIT:     Engage Procedure Spec.                                --
--| Effects:  Determines if Rocket is to be launched, and at what target.-
--| Modifies: No global data is modified.                           --
--| Requires: Status package must set mode and airborne counts.     --
--| Raises:   No explicitly raised exceptions are propagated.       --
--| Engineer: M. Sperry.                                            --
---------------------------------------------------------------------

--
--  The ENGAGE procedure determines if a new rocket should be launched,
--  and if so, which one it should be.  This is based on the MODE, either
--  Manual or Automatic, the number of rockets already in flight, and
--  (when in Manual mode) the LAUNCH button on the operator console.
--  This routine is called during every rocket control task iteration.
--
--  Returned TARGET parameter is ZERO if no target should be engaged,
--  otherwise it indicates the selected target.

with Target;

function Engage(TARGET_INFO : in  Target.TARGET_DATA_LIST_TYPE) return
                Target.TARGET_ID_TYPE;
```

```
-----------------------------------------------------------------------------
--| UNIT:     Engage Procedure Body.                                       --
--| Effects:  Determines if Rocket is to be launched, and at what target.-
--| Modifies: No global data is modified.                                  --
--| Requires: Status package must set mode and airborne counts.            --
--| Raises:   No explicitly raised exceptions are propagated.              --
--| Engineer: M. Sperry.                                                   --
-----------------------------------------------------------------------------


--
--  The ENGAGE procedure determines if a new rocket should be launched,
--  and if so, which one it should be.  This is based on the MODE, either
--  Manual or Automatic, the number of rockets already in flight, and
--  (when in Manual mode) the LAUNCH button on the operator console.
--  This routine is called during every rocket control task iteration.
--
--  Returned TARGET parameter is ZERO if no target should be engaged,
--  otherwise it indicates the selected target.

with Interrupt_Control;
with Status;
with Mouse_Buffer;
with Types;
with Config;
with Shapes;
with Time_Stamp;
        pragma ELABORATE(Interrupt_Control, Status, Mouse_Buffer, Time_Stamp);

function Engage(TARGET_INFO : in  Target.TARGET_DATA_LIST_TYPE) return
                Target.TARGET_ID_TYPE is

use Types;      -- for operators
use Status;     -- for operators

RETICLE_X_PIXEL : Types.WORD;               -- reticle in PIXEL coordinates
RETICLE_Y_PIXEL : Types.WORD;               -- reticle in PIXEL coordinates
RETICLE_X_GRID  : Types.METERS;             -- reticle in GRID coordinates
RETICLE_Y_GRID  : Types.METERS;             -- reticle in GRID coordinates
PREV_DISTANCE   : Types.METERS;
DISTANCE_X      : Types.METERS;
DISTANCE_Y      : Types.METERS := Config.meters_in_battle_area;
TOTAL_DISTANCE  : Types.METERS;
TARGET_ID       : Target.TARGET_ID_TYPE;

begin
  --$TP(0018) Engage start
  TARGET_ID := 0;                              -- default
  if Status.STATUS_CONTROL(Status.AIRBORNE).DATA < Config.max_rockets then
    if Status.MODE = Status.MANUAL then
      if Mouse_Buffer.LAUNCH then
```

```
        -- read ABS_X and ABS_Y in Mouse_Buffer, then convert to METERS types.
        -- Then, find closest target in list to reticle, and give it back.
        Interrupt_Control.Disable;          -- go atomic while reading
        RETICLE_X_PIXEL := Mouse_Buffer.NEW_ABS_X;
        RETICLE_Y_PIXEL := Mouse_Buffer.NEW_ABS_Y;
        Mouse_Buffer.LAUNCH := FALSE;
        Interrupt_Control.Enable;
        RETICLE_X_GRID :=
        Types.METERS(Types.METERS(RETICLE_X_PIXEL -
                    Config.battlefield_screen_left) *
                    Types.METERS(Config.meters_per_X_pixel));
        RETICLE_Y_GRID :=
        Types.METERS(Types.METERS(Config.battlefield_screen_bottom  -
                    RETICLE_Y_PIXEL) *
                    Types.METERS(Config.meters_per_Y_pixel));
        -- This loop locates the closest target to the reticle center
        for ID in Types.TARGET_INDEX_TYPE loop
          if TARGET_INFO(ID).STATUS.ACTIVE and then
             not TARGET_INFO(ID).STATUS.ENGAGED then
            DISTANCE_X := abs(RETICLE_X_GRID - TARGET_INFO(ID).POSITION_NEW.X);
            DISTANCE_Y := abs(RETICLE_Y_GRID - TARGET_INFO(ID).POSITION_NEW.Y);
            if DISTANCE_X <= Shapes.reticle_X_error and
               DISTANCE_Y <= Shapes.reticle_Y_error
            then
              TOTAL_DISTANCE := Types.METERS(DISTANCE_X * DISTANCE_X) +
                                Types.METERS(DISTANCE_Y * DISTANCE_Y);
              if TARGET_ID = 0 or else TOTAL_DISTANCE < PREV_DISTANCE then
                PREV_DISTANCE := TOTAL_DISTANCE;
                TARGET_ID := ID;
              end if;                       -- distance/target check
            end if;                         -- x and y reticle distance check
          end if;                           -- active/not engaged check
        end loop;
      end if;                          -- launch check
    else                        -- automatic mode, search for closest Y value
      for ID in Types.TARGET_INDEX_TYPE loop
        if TARGET_INFO(ID).STATUS.ACTIVE and then
           (not TARGET_INFO(ID).STATUS.ENGAGED and
           TARGET_INFO(ID).POSITION_NEW.Y <= DISTANCE_Y)
        then
          DISTANCE_Y := TARGET_INFO(ID).POSITION_NEW.Y;
          TARGET_ID := ID;
        end if;                 -- active/not engaged/closest y check
      end loop;
    end if;                     -- mode check
  end if;                       -- number of rockets check
  --$TP(0019) Engage end
  return TARGET_ID;
end Engage;
```

```
----------------------------------------------------------------------
--| UNIT:     Graphics Package Spec.                                  --
--| Effects:  Performs all updates to graphics display.               --
--| Modifies: No global data is modified.                             --
--| Requires: Screen must be put in graphics mode by runtime initialize.--
--| Raises:   QUEUE_ERROR is raised if no room for move list.         --
--| Engineer: T. Griest / M. Sperry.                                  --
----------------------------------------------------------------------


-- Date    : 10-10-88
-- Purpose :
--   The Graphics package provides the interface for all screen display
-- operations.  All activity is performed by the Display task which insures
-- that the display is updated in a consistent and timely (c. 100 msecs.)
-- fashion.  The shapes supported are defined in the Shapes package.
-- The initialization in the run-time places the screen in high resolution mode
-- of the EGA's possible types and also sets it to write mode two.  The pixel
-- coordinates passed to the display task are passed as an array of records,
-- where the PIXEL component is in screen coordinates, not meters.  Pixels are
-- defined in high resolution mode as 640 (x) by 350 (y).  Note that the y
-- direction is positive going down.

with Types;
with Config;
with Shapes;

package Graphics is

stack_size       : constant := 8192;             -- in bytes

--
-- define screen and graphics constants
--

subtype COLOR_TYPE is Types.WORD; -- range 0..63;  -- 64 colors on EGA

background_color : constant COLOR_TYPE := 0;     -- black
reticle_color    : constant COLOR_TYPE := 4;     -- red
border_color     : constant COLOR_TYPE := 2;     -- green
status_color     : constant COLOR_TYPE := 7;     -- white
status_box_color : constant COLOR_TYPE := 2;     -- green
rocket_color     : constant COLOR_TYPE := 9;     -- light blue
target_color     : constant array(Types.TARGET_CLASS_TYPE, BOOLEAN) of
        COLOR_TYPE := ((6, 14), (3, 11), (2, 10), (5, 13));
        -- different color for engage = false/true and target type
no_process       : constant COLOR_TYPE := 16;    -- don't process object color

--
-- define graphics data structures
--
```

```
type MOVE_RECORD is record
  XY_OLD          : Shapes.PIXEL;
  XY_NEW          : Shapes.PIXEL;
  OBJECT          : Shapes.OBJECT_TYPE;          -- list of relative offsets
  COLOR           : COLOR_TYPE;         -- if > 15, ignore this motion request
end record;

type MOVE_LIST_TYPE is array (Types.WORD_INDEX range <>) of MOVE_RECORD;
type PRIORITY_TYPE is (HIGH, LOW);

QUEUE_ERROR       : exception;                   -- if queue over/underflow


task type Display_Type is
  entry Move(PRIORITY : PRIORITY_TYPE; WORK_LIST : MOVE_LIST_TYPE);
  pragma PRIORITY(Config.display_priority);
end Display_Type;
for Display_Type'STORAGE_SIZE use INTEGER(Config.bytes_per_storage_unit *
                                                             stack_size);
Display           : Display_Type;

end Graphics;
```

```
-----------------------------------------------------------------------
--| UNIT:     Graphics Package Body                                   --
--| Effects:  Performs all updates to graphics display.               --
--| Modifies: No global data is modified.                             --
--| Requires: Screen must be put in graphics mode by runtime initialize.--
--| Raises:   QUEUE_ERROR is raised if no room for move list.         --
--| Engineer: T. Griest / M. Sperry.                                  --
-----------------------------------------------------------------------


----------------------------------------------------------------------
--      GRAPHICS PACKAGE BODY                                       --
----------------------------------------------------------------------
-- Date    : 10-11-88
-- Purpose :
--   The purpose of the graphics package body is the implementation of the
-- display task on an EGA board.  The display task is responsible for buffering
-- the various tasks that want to draw their particular symbol and the screen.
-- The task begins by waiting for a work request to draw a symbol.  Then, when
-- a request comes in, it is put on a queue.  The queue it goes on is a function
-- of the callers' priority.  Then, since there is work to do, it processes one
-- symbol on the list and rechecks for higher priority work requests until no
-- work on any priority is left.


with Machine_Dependent;
with Interrupt_Control;
with Debug_IO;
with Time_Stamp;
pragma ELABORATE(Machine_Dependent, Interrupt_Control, Debug_IO, Time_Stamp);

package body Graphics is

task body Display_Type is
use Types;                              -- needed for visibility to "+" operator

buffer_size : constant := 256;

type CIRCULAR_BUFFER is array(Types.WORD_INDEX range 0 .. buffer_size - 1) of
                                                           MOVE_RECORD;

type BUFFER_TYPE is record
  ON   : Types.WORD_INDEX := 0;
  OFF  : Types.WORD_INDEX := 0;
  DATA : CIRCULAR_BUFFER;
end record;

SET_PRIORITY : PRIORITY_TYPE := PRIORITY_TYPE'FIRST;
BUFFER       : array(PRIORITY_TYPE'FIRST..PRIORITY_TYPE'LAST) of BUFFER_TYPE; -- set up queues
NO_WORK      : BOOLEAN;                  -- all queues empty?
WORK_REQUEST : MOVE_RECORD;              -- for individual processing
```

```
OBJECT       : Shapes.OBJECT_PTR;        -- current-object to move


procedure Write_To_Screen(WORK_REQUEST : MOVE_RECORD) is

-- A procedure which positions the cursor on the screen (based on the XY_NEW
-- field in the MOVE_RECORD), then prints the string advancing the cursor
-- after each character is printed.  The string is printed character by
-- character to ensure that there are no dependencies on the representation of
-- the type STRING.

begin
  Machine_Dependent.Write_Mode_0;
  Machine_Dependent.Position_Cursor(WORK_REQUEST.XY_NEW.X,
                                    WORK_REQUEST.XY_NEW.Y);
  for I in 1..Config.stats_title_max_length loop
    Machine_Dependent.Write(WORK_REQUEST.OBJECT.TEXT_OBJECT(I),
                            WORK_REQUEST.COLOR);
  end loop;
  Machine_Dependent.Write_Mode_2;
end Write_To_Screen;

procedure Erase_Image(BASE : Shapes.PIXEL;
                      ITEM : Shapes.OBJECT_PTR) is

 - A procedure designed to calculate absolute coordinates for Put_Pixel while
 - placing the pixel in the background color, thus erasing.

begin
  --$TP(0020) Graphics.Erase_Image start
  for I in ITEM.all'range loop
    Machine_Dependent.Put_Pixel(BASE.X + ITEM.all(I).X_OFFSET,
                                BASE.Y + ITEM.all(I).Y_OFFSET,
                                background_color);
  end loop;
  --$TP(0021) Graphics.Erase_Image end
end Erase_Image;
pragma INLINE(ERASE_IMAGE);


procedure Draw_Image(BASE  : Shapes.PIXEL;
                     ITEM  : Shapes.OBJECT_PTR;
                     COLOR : COLOR_TYPE) is

-- A procedure designed to calculate absolute coordinates for Put_Pixel while
-- turning on the pixel in the given color.

begin
  --$TP(0022) Graphics.Draw_Image start
  for I in ITEM.all'range loop
    Machine_Dependent.Put_Pixel(BASE.X + ITEM.all(I).X_OFFSET,
                                    BASE.Y + ITEM.all(I).Y_OFFSET,
                                    COLOR);
  end loop;
  --$TP(0023) Graphics.Draw_Image end
end Draw_Image;
pragma INLINE(DRAW_IMAGE);


procedure Initialize_Border is

-- A procedure used to place a color border around the battlefield limits.

BORDER : MOVE_RECORD;

begin
  BORDER.OBJECT.PIXEL_OBJECT := Shapes.DOT;
  OBJECT := Shapes.OBJECT_PTR_TABLE(BORDER.OBJECT.PIXEL_OBJECT);
  BORDER.COLOR := border_color;
--
-- draw top and bottom border
--
  for I in Config.border_left..Config.border_right loop
    BORDER.XY_NEW := (Types.COORDINATE(I),Config.border_top);
    Draw_Image(BORDER.XY_NEW,OBJECT,BORDER.COLOR);
    BORDER.XY_NEW := (Types.COORDINATE(I),Config.border_bottom);
    Draw_Image(BORDER.XY_NEW,OBJECT,BORDER.COLOR);
  end loop;
--
-- draw left side and right side border
--
  for J in Config.border_top..Config.border_bottom loop
    BORDER.XY_NEW := (Config.border_left,Types.COORDINATE(J));
    Draw_Image(BORDER.XY_NEW,OBJECT,BORDER.COLOR);
    BORDER.XY_NEW := (Config.border_right,Types.COORDINATE(J));
    Draw_Image(BORDER.XY_NEW,OBJECT,BORDER.COLOR);
  end loop;
exception
  when others => Debug_IO.Put_Line("Exception raised in Graphics.Initialize");
end Initialize_Border;


procedure Enqueue(PRIORITY : PRIORITY_TYPE; MOVE_REQUEST : MOVE_RECORD) is

-- A procedure which enqueues a MOVE_RECORD onto the proper priority queue for
-- later processing.  May raise QUEUE_ERROR.

ON_NEW   : Types.WORD_INDEX;

begin
  --$TP(0024) Graphics.Enqueue start
  ON_NEW := (BUFFER(PRIORITY).ON + 1) rem buffer_size;
  if ON_NEW = BUFFER(PRIORITY).OFF then
    raise QUEUE_ERROR;
  end if;
  Interrupt_Control.Disable;                      -- compiler bug
  BUFFER(PRIORITY).DATA(ON_NEW) := MOVE_REQUEST;
  Interrupt_Control.Enable;                       --
  BUFFER(PRIORITY).ON := ON_NEW;
  --$TP(0025) Graphics.Enqueue end
end Enqueue;
pragma INLINE(Enqueue);


procedure Dequeue(PRIORITY : PRIORITY_TYPE; MOVE_REQUEST : out MOVE_RECORD) is

-- A procedure to remove a MOVE_RECORD for processing.  May raise QUEUE_ERROR.

OFF_NEW : Types.WORD_INDEX;

begin
  --$TP(0026) Graphics.Dequeue start
  if BUFFER(PRIORITY).OFF = BUFFER(PRIORITY).ON then
    raise QUEUE_ERROR;
  end if;
  OFF_NEW := (BUFFER(PRIORITY).OFF + 1) rem buffer_size;
  Interrupt_Control.Disable;                      -- compiler bug
  MOVE_REQUEST := BUFFER(PRIORITY).DATA(OFF_NEW);
  Interrupt_Control.Enable;                       --
  BUFFER(PRIORITY).OFF := OFF_NEW;
  --$TP(0027) Graphics.Dequeue end
end Dequeue;
pragma INLINE(Dequeue);


-----------------------------------
--  Body of DISPLAY TASK         --
-----------------------------------
begin
  NO_WORK := TRUE;
  Machine_Dependent.Initialize_Screen;           -- hi-res graphics
  Machine_Dependent.Write_Mode_2;                -- go to write mode 2
  Initialize_Border;                             -- draw battlefield border
  loop
    begin                                        -- exception block
      --$TP(0028) Graphics task start
      if NO_WORK or Move'COUNT > 0 then
        --$TP(0112) Graphics accept Move start
        accept Move(PRIORITY : PRIORITY_TYPE; WORK_LIST : MOVE_LIST_TYPE) do
          for I in WORK_LIST'range loop
            if WORK_LIST(I).COLOR <= 15 then     -- process this symbol?
              Enqueue(PRIORITY, WORK_LIST(I));
            end if;
          end loop;
        end Move;
        --$TP(0113) Graphics accept Move end
        NO_WORK := FALSE;
      end if;
--
--  Now there is some work to do, see if any left on highest priority
--
      SET_PRIORITY := PRIORITY_TYPE'FIRST;
      loop
        if BUFFER(SET_PRIORITY).ON /= BUFFER(SET_PRIORITY).OFF then
          Dequeue(SET_PRIORITY,WORK_REQUEST); -- at this point, requests real
          case WORK_REQUEST.OBJECT.OBJECT_MODE is
            when Shapes.TEXT_MODE  => Write_To_Screen(WORK_REQUEST);
            when Shapes.PIXEL_MODE =>
            OBJECT := Shapes.OBJECT_PTR_TABLE(WORK_REQUEST.OBJECT.PIXEL_OBJECT);
            Erase_Image(WORK_REQUEST.XY_OLD, OBJECT);
            Draw_Image (WORK_REQUEST.XY_NEW, OBJECT, WORK_REQUEST.COLOR);
          end case;
          NO_WORK := FALSE;
          exit;                       -- leave loop if we processed a request
        else
          NO_WORK := TRUE;            -- default
          exit when SET_PRIORITY = PRIORITY_TYPE'LAST;
          SET_PRIORITY := PRIORITY_TYPE'SUCC(SET_PRIORITY);
        end if;
      end loop;

    exception
      when QUEUE_ERROR => null;       -- since error is propagated to caller
      when others =>
        Debug_IO.Put_Line("Error in Display Task");
    end;                              -- exception block
  --$TP(0029) Graphics task end
  end loop;
end Display_Type;


end Graphics;
```

```
---------------------------------------------------------------------------
--| UNIT:     Grid_to_Pixel Function Spec.                                --
--| Effects:  Converts battlefield meters X-Y to graphics Pixel X-Y.      --
--| Modifies: No global data is modified.                                --
--| Requires: No initialization is required.                             --
--| Raises:   No explicitly raised exceptions are propagated.            --
--| Engineer: T. Griest.                                                 --
---------------------------------------------------------------------------
with Shapes;
with Types;
function Grid_to_Pixel(GRID : in Types.POSITION_TYPE) return Shapes.Pixel;
pragma INLINE(Grid_to_Pixel);
```

```
---------------------------------------------------------------------------
--| UNIT:     Grid_to_Pixel Function Spec.                                --
--| Effects:  Converts battlefield meters X-Y to graphics Pixel X-Y.      --
--| Modifies: No global data is modified.                                 --
--| Requires: No initialization is required.                              --
--| Raises:   No explicitly raised exceptions are propagated.             --
--| Engineer: T. Griest.                                                  --
---------------------------------------------------------------------------


with Config;
with Time_Stamp;
pragma ELABORATE(Time_Stamp);
--
--  Translate from Battlefield Grid coordinates in meters to pixels
--  on the screen.  This means applying scale factors for x/y and
--  providing offsets to battlefield area on screen.  NOTE: since
--  battlefield coordinates have 0,0 in lower left; and graphics
--  coordinates have 0,0 in upper left, this involves a transpose of
--  the Y axis (thus the '-').
--
function Grid_to_Pixel(GRID : in Types.POSITION_TYPE) return Shapes.Pixel is
  use Types;
  PIX : Shapes.PIXEL;
begin
  --$TP(0030) Grid_To_Pixel start
  PIX.X := Config.battlefield_screen_left +
           Types.COORDINATE(GRID.X / Types.METERS(Config.meters_per_x_pixel));
  PIX.Y := Config.battlefield_screen_bottom -
           Types.COORDINATE(GRID.Y / Types.METERS(Config.meters_per_y_pixel));
  -- Was previously below return (1-04-89 MPS)
  --$TP(0031) Grid_To_Pixel end
  return PIX;
end Grid_to_Pixel;
```

```
------------------------------------------------------------------
--| UNIT:     Guidance Task Subunit                              --
--| Effects:  Calls "Guide" to compute next rocket aimpoint for every  --
--|           active rocket in the input list.                   --
--| Modifies: No global data is modified.                        --
--| Requires: No initialization is required.                     --
--| Raises:   No explicitly raised exceptions are propagated.    --
--| Engineer: T. Griest.                                         --
------------------------------------------------------------------


with Guide;
with Time_Stamp;
with Interrupt_Control;
pragma ELABORATE(Guide, Time_Stamp, Interrupt_Control);

separate(Rocket)
--
--  Task Type GUIDANCE is used to create an array of tasks which compute
--  guidance information for a specified number of rockets.
--

task body Guidance_Type is

  use Types;              -- for operator visibility
  NEXT_GUIDE_LIST    : AIMPOINT_LIST_TYPE(1..Config.max_rockets);
  NEXT_HISTORY_LIST  : HISTORY_LIST_TYPE(1..Config.max_rockets);
  FIRST_ROCKET_ID    : Types.WORD_INDEX;
  LAST_ROCKET_ID     : Types.WORD_INDEX;

--  Guide computes new aimpoint for rocket based on previous positions
--  function Guide(POS : POSITION_PAIR_TYPE) return Types.AIMPOINT_TYPE
--                                                          is separate;

begin
  loop                         --| main processing loop
    begin                          --| exception block
    --$TP(0032) Guidance task start
--
--  Get history information for our target/rocket list and make local copy.
--  Index of history array is ROCKET_ID.  The entire Guide_List array is
--  used, even though many of the entries may be inactive.  The engagement
--  task is responsible for filtering out only active rockets for issuing
--  the guidance messages.
--
      --$TP(0033) Guidance accept History start
      accept History(HISTORY_DATA : in HISTORY_LIST_TYPE) do
        FIRST_ROCKET_ID := HISTORY_DATA'first;
        LAST_ROCKET_ID  := HISTORY_DATA'last;
        Interrupt_Control.Disable;
        NEXT_HISTORY_LIST(FIRST_ROCKET_ID..LAST_ROCKET_ID) := HISTORY_DATA;
        Interrupt_Control.Enable;
      end History;
      --$TP(0034) Guidance accept History end
--
--  process list to create guidance information
--
      for ROCKET_ID in FIRST_ROCKET_ID..LAST_ROCKET_ID loop
        if NEXT_HISTORY_LIST(ROCKET_ID).ACTIVE then
          NEXT_GUIDE_LIST(ROCKET_ID) :=
                              Guide(NEXT_HISTORY_LIST(ROCKET_ID).POSITION_PAIR);
        end if;
      end loop;

      --$TP(0035) Guidance accept Next_Guidance start
      accept Next_Guidance(AIMPOINT_LIST : out AIMPOINT_LIST_TYPE) do
        if AIMPOINT_LIST'first /= FIRST_ROCKET_ID or
           AIMPOINT_LIST'last  /= LAST_ROCKET_ID
        then
          raise GUIDANCE_LIST_ERROR;
        else
          Interrupt_Control.Disable;
          AIMPOINT_LIST := NEXT_GUIDE_LIST(FIRST_ROCKET_ID..LAST_ROCKET_ID);
          Interrupt_Control.Enable;
        end if;
      end Next_Guidance;
      --$TP(0036) Guidance accept Next_Guidance end

    exception
      when others =>
        Debug_IO.Put_Line("Error in GUIDANCE TASK");
    end;                            --| exception block
  --$TP(0037) Guidance task end
  end loop;                    --| main processing loop
end Guidance_Type;
```

```
---------------------------------------------------------------------
--| UNIT:     Guide Function Spec.                                 --
--| Effects:  Computes a new aimpoint based on rocket/target positions. --
--| Modifies: No global data is modified.                          --
--| Requires: No initialization is required.                       --
--| Raises:   No explicitly raised exceptions are propagated.      --
--| Engineer: T. Griest.                                           --
---------------------------------------------------------------------


with Types;

function Guide(POS : Types.POSITION_PAIR_TYPE) return Types.AIMPOINT_TYPE;
pragma INLINE(Guide);
```

```
------------------------------------------------------------------------
--| UNIT:     Guide Function Body.                                      --
--| Effects:  Computes a new aimpoint based on rocket/target positions. --
--| Modifies: No global data is modified.                               --
--| Requires: No initialization is required.                            --
--| Raises:   No explicitly raised exceptions are propagated.           --
--| Engineer: T. Griest.                                                --
------------------------------------------------------------------------


with Types;
with Math;
with Config;
with Time_Stamp;
pragma ELABORATE(Math, Time_Stamp);
--
--  The Guide function takes the most recent two postions of a rocket/target
--  pair, and computes an aimpoint for the rocket to intercept.
--  Several optimizations are used to provide good average and worst-case
--  performance:
--      1) Rough approximations are used when the rocket is more
--  than a specified closing_time away in any axis;
--      2) rather than using a composite velocity vector, discrete
--  velocities and distances are used for X,Y, and Z to eliminate
--  the need for square root;
--  and a greatly simplified arctan routine is used to approximate (within
--  5 degrees of accuracy) the arctan function using a simple multiply.
--  This also simultaneously converts the fixed point type to BAMs (see Types).
--

function Guide(POS : Types.POSITION_PAIR_TYPE) return Types.AIMPOINT_TYPE is
  use Types;   -- for operators

  climb_Point      : constant := 500.0;  -- climb to 500 meters altitude first
  control_time     : constant := 20;     -- last 20 intervals (2 seconds)

  type AXIS_TYPE is (X, Y , Z);

  MAX_AXIS         : AXIS_TYPE;
  MAX_DIST         : Types.METERS;

  type POSITION_ARRAY_TYPE is array(AXIS_TYPE) of Types.METERS;

  TARGET_POS       : POSITION_ARRAY_TYPE;

  ROCKET_DELTA     : POSITION_ARRAY_TYPE;
  TARGET_DELTA     : POSITION_ARRAY_TYPE;
  AXIS_DIST        : POSITION_ARRAY_TYPE;

  DIST_XY          : Types.METERS;    -- used to project Z on X-Y
  SQ_X             : Types.LONG_FIXED;  -- for squaring
  SQ_Y              : Types.LONG_FIXED;
  SQ_XY             : Types.LONG_FIXED;

  NEXT_AIMPOINT     : Types.AIMPOINT_TYPE;

  CLOSING_RATE      : Types.METERS;           -- METERS per interval
  CLOSING_TIME      : Types.WORD;             -- number of intervals
  MAX_CLOSING_TIME  : Types.WORD;             -- worst case num of intervals

begin
  --$TP(0038) Guide start
--
-- First determine if rocket is in boost phase (initial climb to altitude),
-- if so, simply maintain climb.  NOTE: This technique assumes that the
-- rocket will never fly straight and level below the climb_point.  This
-- implies that the climb_point is sufficiently high that the ingress phase
-- has a constant downward slope.  If this turns out not to be the case,
-- the rocket will jump up to altitude unexpectantly when trying to fly
-- level below the climb_point.  The expected scenario prevents this
-- from happening.  However, to make this more robust, a time_of_flight
-- could be maintained which would determine when to complete the boost
-- phase.
--
  ROCKET_DELTA(Z) := POS.ROCKET_NEW.Z - POS.ROCKET_OLD.Z;  -- rocket change in Z
  if ROCKET_DELTA(Z) >= 0.0 and POS.ROCKET_NEW.Z < climb_point
  then  -- still in boost
    NEXT_AIMPOINT := (Config.launch_azimuth, Config.launch_elevation);
  else                           -- must do some guidance
--
--  Now determine how far away the target is in each axis
--
    AXIS_DIST(X) := POS.TARGET_NEW.X - POS.ROCKET_NEW.X;     -- target/rocket X
    AXIS_DIST(Y) := POS.TARGET_NEW.Y - POS.ROCKET_NEW.Y;     -- target/rocket Y
    AXIS_DIST(Z) := POS.TARGET_NEW.Z - POS.ROCKET_NEW.Z;     -- target/rocket Z

    ROCKET_DELTA(X) := POS.ROCKET_NEW.X - POS.ROCKET_OLD.X; -- rocket change X
    ROCKET_DELTA(Y) := POS.ROCKET_NEW.Y - POS.ROCKET_OLD.Y; -- rocket change Y

    TARGET_DELTA(X) := POS.TARGET_NEW.X - POS.TARGET_OLD.X; -- TARGET change X
    TARGET_DELTA(Y) := POS.TARGET_NEW.Y - POS.TARGET_OLD.Y; -- TARGET change Y
    TARGET_DELTA(Z) := POS.TARGET_NEW.Z - POS.TARGET_OLD.Z; -- TARGET change Z
--
--  Compute the farthest distance "MAX_DIST" and the closing rate for that
--  axis.
--
    MAX_CLOSING_TIME := -1;
    for AXIS in AXIS_TYPE loop
      CLOSING_RATE := ROCKET_DELTA(AXIS)-TARGET_DELTA(AXIS);
--  make sure next calculation will not overflow METER type
      if CLOSING_RATE /= 0.0 and abs AXIS_DIST(AXIS) < 500.0 then
        CLOSING_TIME := Types.WORD(AXIS_DIST(AXIS) / CLOSING_RATE);
        if CLOSING_TIME < 0 then
          MAX_CLOSING_TIME := Types.WORD'last;
        else
          if CLOSING_TIME > MAX_CLOSING_TIME then
            MAX_CLOSING_TIME := CLOSING_TIME;
            MAX_AXIS := AXIS;
          end if;
        end if;
      end if;
    end loop;

--
--  Compute number of intervals before target/rocket intercept.
--  Note: this operation's accuracy depends on the rounding algorithm used...
--  If the Rocket is close (in time) to the Rocket, do extra work
--  of extrapolating the Targets position at intercept
--
    if MAX_CLOSING_TIME > 0 and
      MAX_CLOSING_TIME < control_time then   -- extrapolate
      AXIS_DIST(X) := AXIS_DIST(X) + TARGET_DELTA(X)*INTEGER(MAX_CLOSING_TIME);
      AXIS_DIST(Y) := AXIS_DIST(Y) + TARGET_DELTA(Y)*INTEGER(MAX_CLOSING_TIME);
      AXIS_DIST(Z) := AXIS_DIST(Z) + TARGET_DELTA(Z)*INTEGER(MAX_CLOSING_TIME);
    end if;
--
--  Now compute angle for Azimuth and Elevation to
--  relative target position (possibly extrapolated) for this rocket.
--
    NEXT_AIMPOINT.AZIMUTH   := Math.Arctan(AXIS_DIST(X),AXIS_DIST(Y));
    SQ_X := Types.LONG_FIXED(AXIS_DIST(X));
    SQ_X := Types.LONG_FIXED(SQ_X * SQ_X);
    SQ_Y := Types.LONG_FIXED(AXIS_DIST(Y));
    SQ_Y := Types.LONG_FIXED(SQ_Y * SQ_Y);
    SQ_XY := Math.Sqrt(SQ_X + SQ_Y);
    if SQ_XY > 4000.0 then
      NEXT_AIMPOINT.ELEVATION := -200;--fly almost level till rocket gets closer
    else
      DIST_XY := Types.METERS(SQ_XY);
      NEXT_AIMPOINT.ELEVATION := Math.Arctan(DIST_XY,AXIS_DIST(Z));
      if NEXT_AIMPOINT.ELEVATION > -1024 then
        NEXT_AIMPOINT.ELEVATION := -1024;
      elsif NEXT_AIMPOINT.ELEVATION < -16384 then
        NEXT_AIMPOINT.ELEVATION := -16384;
      end if;          -- elevation check
    end if;            -- SQ_XY too big check
  end if;              -- no longer in boost check
  --STP(0039) Guide end
  return NEXT_AIMPOINT;
end Guide;
```

```
--------------------------------------------------------------------------
--| UNIT:     Guide_Buf Task Subunit                                     --
--| Effects:  Provides asynchronous comm. between simulator and Control.--
--| Modifies: No global data is modified.                                --
--| Requires: No initialization is required.                             --
--| Raises:   No explicitly raised exceptions are propagated.            --
--| Engineer: T. Griest.                                                 --
--------------------------------------------------------------------------


--
--  The Guide_Buf task acts as a buffer between the rocket data link
--  support task Rock_Sup and the Rocket.Control task which processes
--  the rocket data.
--
--  TIMING CONSIDERATIONS:  Guide_Buf will only provide the most recent
--  message received.  If two messages are received prior to one being
--  taken, the first will be lost.
--
with Debug_IO;
with Time_Stamp;
pragma ELABORATE(Debug_IO, Time_Stamp);

separate (Simulate.RDL)

task body Guide_Buf_Type is

  use Types;
  GUIDE_MSG    : Rocket.ROCKET_GUIDE_MSG_TYPE;
  MSG_COUNT    : Types.WORD := 0;  -- if a message has been buffered
begin
  loop
    --$TP(0040) Guidebuf task start
    select
      accept Put_Guide(DATA : in Rocket.ROCKET_GUIDE_MSG_TYPE) do
        --$TP(0041) Guidebuf accept Put_Guide start
        GUIDE_MSG.NUM_ROCKETS := DATA.NUM_ROCKETS;          -- copy data
        for I in Types.WORD_INDEX range 1..DATA.NUM_ROCKETS loop
          GUIDE_MSG.ROCKET_GUIDE_LIST(I) := DATA.ROCKET_GUIDE_LIST(I);
        end loop;
        MSG_COUNT := 1;          -- only meaningful that it is > 0
        --$TP(0042) Guidebuf accept Put_Guide end
      end Put_Guide;
    or
      when MSG_COUNT > 0 =>
      accept Get_Guide(DATA : out Rocket.ROCKET_GUIDE_MSG_TYPE) do
        --$TP(0043) Guidebuf accept Get_Guide start
        DATA.NUM_ROCKETS := GUIDE_MSG.NUM_ROCKETS;
        for I in Types.WORD_INDEX range 1..GUIDE_MSG.NUM_ROCKETS loop
          DATA.ROCKET_GUIDE_LIST(I) := GUIDE_MSG.ROCKET_GUIDE_LIST(I);
        end loop;
        MSG_COUNT := 1;             -- do keep multiple copies
        --$TP(0044) Guidebuf accept Get_Guide end
      end Get_Guide;
    end select;
  --$TP(0045) Guidebuf task end
  end loop;
exception
  when others =>
    Debug_IO.Put_Line("GUIDE_BUF termination due to exception.");
end Guide_Buf_Type;
```

```
---------------------------------------------------------------------------
--| UNIT:     Interrupt_Control Package Spec. and Body.                  --
--| Effects:  Provides control over interrupt flags.                     --
--| Modifies: No global data is modified.                                --
--| Requires: No initialization is required.                             --
--| Raises:   No explicitly raised exceptions are propagated.            --
--| Engineer: M. Sperry.                                                 --
---------------------------------------------------------------------------


-- Date    : 11-09-88
-- Purpose :
--   The purpose of the Interrupt_Control package is to provide Ada level
-- semantics for disabling and enabling interrupts on the 80X86 family of
-- processors.  Also for clearing the direction flag because of an RTE bug
-- which does not always clear it.

with Machine_Code;
use Machine_Code;
pragma ELABORATE(Machine_Code);

package Interrupt_Control is

pragma SUPPRESS(Elaboration_Check);

procedure Disable;
pragma INLINE(Disable);
procedure Enable;
pragma INLINE(Enable);

procedure Clear_Direction_Flag;
pragma INLINE(Clear_Direction_Flag);

end Interrupt_Control;

package body Interrupt_Control is

procedure Disable is
begin
  MACHINE_INSTRUCTION'(none,m_CLI);
end Disable;

procedure Enable is
begin
  MACHINE_INSTRUCTION'(none,m_STI);
end Enable;

procedure Clear_Direction_Flag is
begin
  MACHINE_INSTRUCTION'(none,m_CLD);
end Clear_Direction_Flag;
end Interrupt_Control;
```

```
-------------------------------------------------------------------
--| UNIT:     Machine_Dependent Package Spec.                     --
--| Effects:  Provides graphics machine dependencies.             --
--| Modifies: No global data is modified.                         --
--| Requires: No initialization is required (other than graphics mode). --
--| Raises:   No explicitly raised exceptions are propagated.     --
--| Engineer: M. Sperry.                                          --
-------------------------------------------------------------------


-- Date    : 11-04-88
-- Purpose :
--   The purpose of the Machine_Dependent package is to provide a separate
-- package from the Ada code for drawing on an EGA high resolution screen via
-- code statements.


with Machine_Code;
with Graphics;
with Types;
use Machine_Code;
pragma ELABORATE(Machine_Code);

package Machine_Dependent is

procedure Put_Pixel(ABS_X, ABS_Y : Types.COORDINATE;
                    COLOR        : Graphics.COLOR_TYPE);
pragma INLINE(Put_Pixel);

procedure Initialize_Screen;
pragma INLINE(Initialize_Screen);

procedure Write_Mode_0;
pragma INLINE(Write_Mode_0);

procedure Position_Cursor(X,Y : Types.COORDINATE);
pragma INLINE(Position_Cursor);

procedure Write(CHAR  : CHARACTER;
                COLOR : Graphics.COLOR_TYPE);
pragma INLINE(Write);

procedure Write_Mode_2;
pragma INLINE(Write_Mode_2);

end Machine_Dependent;
```

```
------------------------------------------------------------------
--| UNIT:     Machine_Dependent Package Body.                      --
--| Effects:  Provides graphics machine dependencies.             --
--| Modifies: No global data is modified.                         --
--| Requires: No initialization is required (other than graphics mode). --
--| Raises:   No explicitly raised exceptions are propagated.     --
--| Engineer: M. Sperry.                                          --
------------------------------------------------------------------

package body Machine_Dependent is

-- A machine dependent package which makes use of the functionality of the
-- Phoenix BIOS routines to perform some graphics processing.  The BIOS call
-- used is at C000:0CD7 (which is INT 10 on most EGA adapters).

hi_res_graphics  : constant := 16#10#;          -- graphics mode
set_cursor       : constant := 16#0200#;        -- set cursor function
page_zero        : constant := 16#00#;          -- set cursor to active page
write_function   : constant := 16#0E#;
index_register   : constant := 16#3CE#;         -- port address
access_register  : constant := 16#3CF#;         -- port address
mode_register    : constant := 5;               -- index register 5
write_mode_2_val : constant := 2;
write_mode_0_val : constant := 0;


procedure Put_Pixel(ABS_X, ABS_Y : Types.COORDINATE;
                    COLOR        : Graphics.COLOR_TYPE) is

-- An assembly level procedure (for enhanced speed) to place a dot on the EGA
-- screen.  Write mode two is used here (again, for enhanced speed).  It is
-- important to note that this routine is called more frequently that any
-- other.

begin
--
-- The first thing to do is find out which bit must be turned on.  This is
-- done by taking SHR( 80h, ABS_X mod 8 ).  The bit ordering goes from 7 -> 0.
--
  MACHINE_INSTRUCTION'(register_register, m_MOV, CX, CX); -- defeat compiler bug

  MACHINE_INSTRUCTION'(register_immediate, m_MOV, DX, 16#3CE#);  -- select bit
  MACHINE_INSTRUCTION'(register_immediate, m_MOV, AL, 8);  -- mask register
  MACHINE_INSTRUCTION'(register_register, m_OUT, DX, AL);  -- in graphics chip
--
-- Determine which bit must be turned on.  This is
-- done by taking SHR( 80h, ABS_X rem 8 ), reversing the bit ordering.
--
  MACHINE_INSTRUCTION'(register_system_address, m_MOV, CX, ABS_X'address); --X
  MACHINE_INSTRUCTION'(register_register, m_MOV, BX, CX);    -- make copy of X
  MACHINE_INSTRUCTION'(register_immediate, m_AND, CL, 7);    -- mask for bit #
  MACHINE_INSTRUCTION'(register_immediate, m_MOV, AL, 16#80#);  -- most significant bit is
  MACHINE_INSTRUCTION'(register_register, m_SHR, AL, CL); -- bit zero, do bit reversal.
--
-- AL now holds the bit mask.  Now give it to the bit mask register located
-- at 16#3CF#.
--
  MACHINE_INSTRUCTION'(register, m_INC, DX);  -- increment port address to 3CF
  MACHINE_INSTRUCTION'(register_register, m_OUT, DX, AL);


--
-- Now, latch the byte of graphics memory.  The byte to latch
-- is defined as (ABS_Y * 80) + (ABS_X / 8).  Then, when giving
-- it back, place the color in AL.  Note that only four bits of the color are
-- significant and that the color placed in AL is not actually a color, but a
-- palette register selection (from 0 to 15).  The color in the palette
-- register is the color displayed.-16#6000# is loaded (= A000H)
-- to point to the EGA graphics page zero memory address.
--
  MACHINE_INSTRUCTION'(register_system_address, m_MOV, AX, ABS_Y'address);
  MACHINE_INSTRUCTION'(register_immediate, m_MOV, CX, 80); -- bytes/line
  MACHINE_INSTRUCTION'(register, m_MUL, CX);      -- ABS_Y * 80 in AX
  MACHINE_INSTRUCTION'(register_immediate, m_MOV, CL, 3);  -- Shift Count
  MACHINE_INSTRUCTION'(register_register, m_SHR, BX, CL);  -- ABS_X / 8 in BX
  MACHINE_INSTRUCTION'(register_register, m_ADD, BX, AX);  -- BX is offset
  MACHINE_INSTRUCTION'(register_immediate, m_MOV, AX, -16#6000#); -- base of RAM
  MACHINE_INSTRUCTION'(register_register, m_MOV, ES, AX);
--
-- Latch the palette selection.  Note that the contents of AL upon return are
-- meaningless, and that the color is latched internally to the EGA's four bit
-- planes.
--
  MACHINE_INSTRUCTION'(register_address, m_MOV, AL, ES, BX, 0);-- mov al,es:[bx]
  MACHINE_INSTRUCTION'(register_system_address, m_MOV, AX, COLOR'address);
--
-- Finally, give the palette selection (color) to the four bit planes.
--
  MACHINE_INSTRUCTION'(address_register, m_MOV, ES, BX, 0, AL);-- mov es:[bx],al

end Put_Pixel;


--
--  Provide a mechanism to call ROM located routine to initialize screen
--
procedure Int10;    -- spec of interface to BIOS graphics call
pragma INTERFACE(ASM86, Int10);
pragma INTERFACE_SPELLING(Int10, "D1BIOS?GRAPHICSCALL");


procedure Initialize_Screen is
-- A procedure used to place an EGA screen into mode 10h, which is 640 x 350
-- pixels.  The screen is initially in write mode 0, which is different from
-- the graphics mode 10h.  Later, in Change_To_Mode_2, the write mode is
-- changed to mode 2.

begin
  MACHINE_INSTRUCTION'(register_immediate, m_MOV, AX, hi_res_graphics);
  MACHINE_INSTRUCTION'(none, m_PUSHF);
  MACHINE_INSTRUCTION'(register_system_address, m_CALL, Int10'Address);
end Initialize_Screen;


procedure Write_Mode_0 is
-- A procedure used to change the write mode of the screen to mode 0, for text
-- writing.  This procedure is called before writing the necessary statistics
-- titles.

begin
  MACHINE_INSTRUCTION'(register_immediate, m_MOV, DX, index_register);
  MACHINE_INSTRUCTION'(register_immediate, m_MOV, AL, mode_register);
  MACHINE_INSTRUCTION'(register_register, m_OUT, DX, AL);
  MACHINE_INSTRUCTION'(register_immediate, m_MOV, DX, access_register);
  MACHINE_INSTRUCTION'(register_immediate, m_MOV, AL, write_mode_0_val);
  MACHINE_INSTRUCTION'(register_register, m_OUT, DX, AL);
end Write_Mode_0;


procedure Position_Cursor(X,Y : Types.COORDINATE) is
-- A procedure used to place the cursor where necessary (procedure Write
-- only writes at the current cursor position).

begin
  MACHINE_INSTRUCTION'(register_immediate, m_MOV, AX, set_cursor);
  MACHINE_INSTRUCTION'(register_system_address, m_MOV, DL, X'address);
  MACHINE_INSTRUCTION'(register_system_address, m_MOV, DH, Y'address);
  MACHINE_INSTRUCTION'(register_immediate, m_MOV, BX, page_zero);
  MACHINE_INSTRUCTION'(none, m_PUSHF);
  MACHINE_INSTRUCTION'(register_system_address, m_CALL, Int10'Address);
end Position_Cursor;


procedure Write(CHAR  : CHARACTER;
                COLOR : Graphics.COLOR_TYPE) is
-- A procedure used to place the character CHAR at the current cursor position
-- on the screen which is in hi-res graphics mode, write mode 0.  This procedure
-- uses function call 0eh, which interprets ascii codes 10 and 13 as linefeed
-- and carriage return respectively.  The cursor can then be controlled so that
-- a character can be placed anywhere on the screen.

begin
  MACHINE_INSTRUCTION'(register_immediate, m_MOV, AH, write_function);
  MACHINE_INSTRUCTION'(register_system_address, m_MOV, AL, CHAR'address);
  MACHINE_INSTRUCTION'(register_system_address, m_MOV, BL, COLOR'address);
  MACHINE_INSTRUCTION'(none, m_PUSHF);
  MACHINE_INSTRUCTION'(register_system_address,m_CALL, Int10'address);
end Write;



procedure Write_Mode_2 is
-- A procedure used to change the write mode of the screen to mode 2, for pixel
-- plotting.  This procedure is called after writing the necessary statistics
-- titles.

begin
  MACHINE_INSTRUCTION'(register_immediate, m_MOV, DX, index_register);
  MACHINE_INSTRUCTION'(register_immediate, m_MOV, AL, mode_register);
  MACHINE_INSTRUCTION'(register_register, m_OUT, DX, AL);
  MACHINE_INSTRUCTION'(registe-_immediate, m_MOV, DX, access_register);
  MACHINE_INSTRUCTION'(register_immediate, m_MOV, AL, write_mode_2_val);
  MACHINE_INSTRUCTION'(register_register, m_OUT, DX, AL);
end Write_Mode_2;

end Machine_Dependent;
```

```
--------------------------------------------------------------------
--| UNIT:     Math Package Spec.                                    --
--| Effects:  Compute various functions: Tan, Arc Tan, and Sqrt.    --
--| Modifies: No global data is modified.                           --
--| Requires: No initialization is required.                        --
--| Raises:   No explicitly raised exceptions are propagated.       --
--| Engineer: T. Griest.                                            --
--------------------------------------------------------------------

with Types;

package Math is

function Tan (ANGLE : Types.BAM) return Types.LONG_FIXED;

function Sqrt(X : in Types.METERS) return Types.METERS;

function Sqrt(X : in Types.LONG_FIXED) return Types.LONG_FIXED;

function Arctan(REL_X, REL_Y : in Types.METERS) return Types.BAM;

end Math;
```

```
----------------------------------------------------------------------
--| UNIT:      Math Package Body.                                    --
--| Effects:  Compute various functions: Tan, Arc Tan, and Sqrt.     --
--| Modifies: No global data is modified.                            --
--| Requires: No initialization is required.                         --
--| Raises:   No explicitly raised exceptions are propagated.        --
--| Engineer: T. Griest.                                             --
----------------------------------------------------------------------

with Time_Stamp;
pragma ELABORATE(Time_Stamp);

package body Math is
  use Types;

  TAN_TABLE  : array(Types.WORD range 0..90) of Types.LONG_FIXED :=
(
 0.00000,  0.01746,  0.03492,  0.05241,  0.06993,  0.08749,  0.10510,  0.12278,
 0.14054,  0.15838,  0.17633,  0.19438,  0.21256,  0.23087,  0.24933,  0.26795,
 0.28675,  0.30573,  0.32492,  0.34433,  0.36397,  0.38386,  0.40403,  0.42447,
 0.44523,  0.46631,  0.48773,  0.50953,  0.53171,  0.55431,  0.57735,  0.60086,
 0.62487,  0.64941,  0.67451,  0.70021,  0.72654,  0.75356,  0.78129,  0.80978,
 0.83910,  0.86929,  0.90040,  0.93252,  0.96569,  1.00000,  1.03553,  1.07237,
 1.11061,  1.15037,  1.19175,  1.23490,  1.27994,  1.32704,  1.37638,  1.42815,
 1.48256,  1.53986,  1.60033,  1.66428,  1.73205,  1.80405,  1.88073,  1.96261,
 2.05030,  2.14451,  2.24604,  2.35585,  2.47509,  2.60509,  2.74748,  2.90421,
 3.07768,  3.27085,  3.48741,  3.73205,  4.01078,  4.33148,  4.70463,  5.14455,
 5.67128,  6.31375,  7.11536,  8.14434,  9.51436, 11.43005, 14.30067, 19.08114,
28.63625, 57.28996, Types.sqrt_large_number);

function Tan (ANGLE : Types.BAM) return Types.LONG_FIXED is
  TANGENT      : Types.LONG_FIXED;
  THETA        : Types.WORD;
--  procedure Dummy is begin null; end;   --(UNIX)
begin
  --$TP(0048) Math.Tan start
  THETA := Types.WORD(ANGLE/182);      -- approx. 182 bams per degree
  if THETA >= -90 and THETA <= 90 then
    if THETA >= 0 then
      TANGENT := TAN_TABLE(THETA);
    else
--      Dummy;--(UNIX)
      TANGENT := -TAN_TABLE(-THETA);
    end if;
  elsif THETA < -90 then
    TANGENT := TAN_TABLE(THETA + 180);
  else
    TANGENT := -TAN_TABLE(180-THETA);
  end if;
  --$TP(0049) Math.Tan end
  return TANGENT;
```

```
end Tan;


function Sqrt(X : in Types.METERS) return Types.METERS is
  use Types;   -- import operators
  F : Types.METERS := X;
  Y : Types.METERS := 1.0;
  OLD_Y : Types.METERS := Y;
begin
  --$TP(0050) Math.Sqrt start (METERS)
  for I in 1..15 loop
    exit when Y = 0.0;
    Y := ( Y + Types.METERS(F/Y) ) / 2;
    if Y = OLD_Y then
      exit;
    end if;
    OLD_Y := Y;
  end loop;
  --$TP(0051) Math.Sqrt end (METERS)
  return Y;
end Sqrt;

function Sqrt(X : in Types.LONG_FIXED) return Types.LONG_FIXED is
  use Types;   -- import operators
  F : Types.LONG_FIXED := X;
  Y : Types.LONG_FIXED := 1.0;
  OLD_Y : Types.LONG_FIXED := Y;
begin
  --$TP(0052) Math.Sqrt start (LONG_FIXED)
  for I in 1..15 loop
    exit when Y = 0.0;
    Y := ( Y + Types.LONG_FIXED(F/Y) ) / 2;
    if Y = OLD_Y then
      exit;
    end if;
    OLD_Y := Y;
  end loop;
  --$TP(0053) Math.Sqrt end (LONG_FIXED)
  return Y;
exception
  when NUMERIC_ERROR => Y := OLD_Y;
                        return Y;
end Sqrt;

--
-- A function used to return an approximation of the arctangent function.
-- The maximum error allowed is five degrees off. The Arctan function
-- computes the arctangent handling each quadrant on a case by case basis,
-- since one general algorithm would violate the maximum error condition.
--
```

```
function Arctan(REL_X, REL_Y : in Types.METERS) return Types.BAM is

  type BIG_FIX is delta 0.125 range -200_000_000.0 .. 200_000_000.0;

  offset     : constant Types.BAM := 2 * 182;  -- two degrees offset
  rotate_45  : constant BIG_FIX := 8192.0;
  rotate_90  : constant BIG_FIX := 16384.0;
  rotate_135 : constant BIG_FIX := 24575.0;

  X          : BIG_FIX;
  Y          : BIG_FIX;
  TEMP       : BIG_FIX;
  ANGLE      : Types.BAM;

begin
  --$TP(0054) Math.Arctan start
  X := BIG_FIX(REL_X);
  Y := BIG_FIX(REL_Y);
-- Quadrants:
--                     |
--                II   |   I
--      +/-180  ---------------- 0
--                III  |  IV
--
   if X < 0.0 then                              -- II or III
     X := -X;
     if Y < 0.0 then                            -- III
       Y := -Y;
       if X <= Y then                           -- upper 45 degrees
         X := BIG_FIX(X * 8192 / Y);
         X := rotate_45 - X;
         X := rotate_135 - X;
         ANGLE := Types.BAM(-X) - offset;
       else                                     -- low 45 degrees
         TEMP := X;
         X := Y;                                -- convert
         Y := TEMP;
         X := BIG_FIX(X * 8192/Y);
         ANGLE := Types.BAM(X) + offset + Types.BAM'first;
       end if;
     else                                       -- II
       if X <= Y then                           -- upper 45 degrees
         X := BIG_FIX(X * 8192/Y);
         ANGLE := Types.BAM(X) + offset + 16384;
       else                                     -- low 45 degrees
         TEMP := X;
         X := Y;
         Y := TEMP;
         X := BIG_FIX(X * 8192);
         X := BIG_FIX(X / Y);
         X := rotate_45 - X;
```

```
      X := X + rotate_135;
      ANGLE := Types.BAM(X);
    end if;
  end if;
else                                          -- I or IV
  if Y < 0.0 then                             -- IV
    Y := -Y;
    if X <= Y then                            -- upper 45 degrees
      X := BIG_FIX(X * 8192/Y);
      ANGLE := Types.BAM(X) + offset - 16384;
    else                                      -- low 45 degrees
      TEMP := X;
      X := Y;                                 -- convert
      Y := TEMP;
      X := rotate_45 - BIG_FIX(X * 8192/Y);
      X := X - rotate_45;
      ANGLE := Types.BAM(X) - offset;
    end if;
  else                                        -- I
    if X < Y then                             -- upper 45 degrees
      if Y - X >= X then
        X := rotate_45 + (rotate_45 - BIG_FIX(X * 8192 / Y));
      else
        X := BIG_FIX(X * 8192 / (X + Y));
        X := rotate_45 + X;
      end if;
      ANGLE := Types.BAM(X);
    else                                      -- low 45 degrees
      if Y = 0.0 and X = 0.0 then
        ANGLE := Types.BAM(0);
      else
        TEMP := X;
        X := Y;                               -- convert
        Y := TEMP;
        X := BIG_FIX(X * 8192/Y);
        ANGLE := Types.BAM(X) + offset;
      end if;
    end if;

  end if;
end if;

--$TP(0055) Math.Arctan end
return ANGLE;

end Arctan;

end Math;
```

```
-----------------------------------------------------------------------
--| UNIT:     Mouse Package Spec.                                      --
--| Effects:  Provides graphics pointing device interrupt handling.    --
--| Modifies: Status Mode, and Mouse_Buffer X-Y positions are updated. --
--| Requires: Runtime initialization of interrupt vector.              --
--| Raises:   Task will terminate on MOUSE_ERROR.                      --
--| Engineer: M. Sperry.                                               --
-----------------------------------------------------------------------


-- Date    : 9-30-88
-- Purpose :
--   This is the specification for the package mouse.  In addition to
-- establishing communications with the mouse, a task is provided which
-- handles the receive interrupt generated by the mouse at COM2.

with System;   pragma ELABORATE(System);

package Mouse is


procedure Initialize;

task Char_In is
  pragma INTERRUPT_HANDLER;
  entry REPORT;
  for REPORT use at (16#83#,0);                 -- COM2 8250 serial port vector
end Char_In;

end Mouse;
```

```
----------------------------------------------------------------
--| UNIT:     Mouse Package Body.                              --
--| Effects:  Provides graphics pointing device interrupt handling.   --
--| Modifies: Status Mode, and Mouse_Buffer X-Y positions are updated.  --
--| Requires: Runtime initialization of interrupt vector.         --
--| Raises:   Task will terminate on MOUSE_ERROR.                  --
--| Engineer: M. Sperry.                                          --
----------------------------------------------------------------


-- Date    : 9-30-88
-- Purpose :
--   Package Mouse provides one task and one procedure.  The procedure
-- Initialization sets up the mouse at 4800 baud, no parity, 7 data bits, and
-- two stop bits.  The number of stop bits is insignificant.  There should
-- only be two formats that the mouse can be in, either relative bit pad one
-- or Micrsoft Mouse.  The default on power up for the mouse is MM at 4800.
-- The mouse must be commanded in the following order: BAUD (which is set to
-- default to 4800 so it is not necessary to reprogram it), # of reports/sec.,
-- and then format of the reports.

with Types;
with Low_Level_IO;
with Debug_IO;
with Mouse_Buffer;
with Mouse_Data;                -- provides constants and data structures
with Status;
with Interrupt_Control;
with Time_Stamp;
use Low_Level_IO;
use Mouse_Data;                 -- visibility to "end" function

pragma ELABORATE(Low_Level_IO, Debug_IO, Mouse_Buffer, Status, Time_Stamp);

package body Mouse is

DATA              : Low_Level_IO.BYTE;           -- char from mouse
BUTTON_PUSHED     : Mouse_Data.BIT_FIELD;        -- array representing keys
STATUS_BYTE       : Mouse_Data.BIT_FIELD;        -- represents status errors
PREV_BUTTON_PUSH  : Mouse_Data.BIT_FIELD := (others => FALSE); --previous buttons
MOUSE_INPUT       : Mouse_Data.RAW_MOUSE_WORD := (0,0,0);-- transform to 12-bit

MOUSE_REPORT      : Mouse_Data.SIGNED_MOUSE_WORD; -- transformation to signed
REPORT_COUNT      : Types.WORD range 0..5 := 0;  -- counts byte in report
CHANGE_REQUESTED  : BOOLEAN := FALSE;            -- rendezvous with status?
MOUSE_ERROR       : EXCEPTION;

TEMP_X            : Types.WORD;                  -- local copy of X motion
TEMP_Y            : Types.WORD;                  -- local copy of Y motion

procedure Initialize is
```

```
-- A procedure used to initialize the mouse to 4800 baud and Relative Bit
-- Pad One format.

INTERRUPTS : Low_Level_IO.BYTE;                   -- for input of 8259 ints
RESPONSE   : Low_Level_IO.BYTE;                   -- for mouse responses
TIME_OUT   : INTEGER := 30000;                    -- time out for mouse response

begin
  Receive_Control(Mouse_Data.COM2_status,RESPONSE);  -- clean out junk in status
  Receive_Control(Mouse_Data.COM2_data,RESPONSE);    -- clean out junk in data
  Send_Control(Mouse_Data.COM2_control,Mouse_Data.access_baud);
  Send_Control(Mouse_Data.COM2_data,Mouse_Data.host_baud); -- set BAUD = 4800
-- set COM2 serial parameters
  Send_Control(Mouse_Data.COM2_control,Mouse_Data.host_format);
  Send_Control(Mouse_Data.COM2_data,Mouse_Data.acknowledge); -- touch mouse
  loop
    Receive_Control(Mouse_Data.COM2_status,RESPONSE);   -- wait for response
    if RESPONSE = Mouse_Data.data_new then
      Receive_Control(Mouse_Data.COM2_data,RESPONSE);   -- clear out byte
      exit;
    else
      TIME_OUT := TIME_OUT - 1;
    end if;
    if TIME_OUT = 0 then
      exit;
    end if;
  end loop;
  if TIME_OUT = 0 then
    Debug_IO.Put_Line("Unable to establish communications with mouse.");
  end if;
  Send_Control(Mouse_Data.COM2_data,Mouse_Data.mouse_char_speed);
  delay 0.01;                                   -- slow for mouse input buffer
  Send_Control(Mouse_Data.COM2_data,Mouse_Data.mouse_format);
  Send_Control(Mouse_Data.COM2_modem_control,Mouse_Data.general_int_enable);
  Send_Control(Mouse_Data.COM2_int_enable,Mouse_Data.specific_int_enable);
  Receive_Control(Mouse_Data.pic_8259_mr,INTERRUPTS);
-- enable COM2 in PIC in line below
  INTERRUPTS := Mouse_Data.Bits_to_Byte
           (Mouse_Data.Byte_to_Bits(INTERRUPTS) and Mouse_Data.pic_and_mask);
  Send_Control(Mouse_Data.pic_8259_mr,INTERRUPTS);
end Initialize;


task body Char_In is

-- One of the main tasks used to move the reticle around the battlefield screen.
-- The task rendezvous with the graphics task reporting positions every 28
-- milliseconds, unless the middle button is pressed (mode) changing the mode
-- to AUTOMATIC.  In this event, the mouse simply waits for a change to
-- MANUAL, since automatic mode is controlled by the rocket task.  The mouse
-- task will not rendezvous with the graphics task until set to MANUAL.  When
-- in MANUAL mode, the task (upon completion of one report) will rendezvous
-- with the graphics task at high priority to report it's position.  It will
-- then change the status task's shared variables if any need to be changed.
-- If one does, and the status task has completed it's previous work and gone
-- to an accept state, then the mouse task wakes it up.

use Status;                                   -- for visibility to "="
use Types;                                    -- for visibility to "+"

begin
  loop
    accept Report do
      Interrupt_Control.Clear_Direction_Flag;      -- (RTE bug)
      --$TP(0056) Mouse task start
      Receive_Control(Mouse_Data.COM2_status,DATA); -- receive status
      STATUS_BYTE := Mouse_Data.Byte_to_Bits(DATA);--check statusbyte for errors
      if STATUS_BYTE(Mouse_Data.overflow) or
         STATUS_BYTE(Mouse_Data.framing) then
        REPORT_COUNT := 0;                         -- start a new report
        Receive_Control(Mouse_Data.COM2_data,DATA); -- clear out data port
      else
        Receive_Control(Mouse_Data.COM2_data,DATA); -- get valid data
        if DATA > Mouse_Data.sync_byte then        -- check for new report
          REPORT_COUNT := 1;                      -- start of new report
        end if;
      end if;
      case REPORT_COUNT is                        -- convert data to mouse X,Y
        when 1 =>                                 -- or buttons.
          BUTTON_PUSHED := Mouse_Data.Byte_to_Bits(DATA);
          REPORT_COUNT := REPORT_COUNT + 1;
        when 2 =>
          MOUSE_INPUT.LOW := Mouse_Data.Byte_to_Bit6(DATA);
          REPORT_COUNT := REPORT_COUNT + 1;
        when 3 =>
          MOUSE_INPUT.HIGH := Mouse_Data.Byte_to_Bit6(DATA);
          MOUSE_REPORT := Mouse_Data.Raw_to_Signed(MOUSE_INPUT);
          TEMP_X := MOUSE_REPORT.LOW12;
          REPORT_COUNT := REPORT_COUNT + 1;
        when 4 =>
          MOUSE_INPUT.LOW := Mouse_Data.Byte_to_Bit6(DATA);
          REPORT_COUNT := REPORT_COUNT + 1;
        when 5 =>
          -- don't move mouse if any buttons pushed.
          if (not BUTTON_PUSHED(Mouse_Data.reset)) and -- guarantee only one -
             (not BUTTON_PUSHED(Mouse_Data.mode)) and  -- rendezvous per report-
             (not BUTTON_PUSHED(Mouse_Data.launch)) then  -- (RTE bug) -
            PREV_BUTTON_PUSH(Mouse_Data.reset)  := FALSE;
            PREV_BUTTON_PUSH(Mouse_Data.mode)   := FALSE;
            PREV_BUTTON_PUSH(Mouse_Data.launch) := FALSE;
            MOUSE_INPUT.HIGH := Mouse_Data.Byte_to_Bit6(DATA);
  MOUSE_REPORT := Mouse_Data.Raw_to_Signed(MOUSE_INPUT);
  TEMP_Y := MOUSE_REPORT.LOW12;
  if Status.MODE = Status.MANUAL then
    MOUSE_BUFFER.MOUSE_X := TEMP_X;
    MOUSE_BUFFER.MOUSE_Y := TEMP_Y;
    --$TP(0057) Mouse rendezvous with Save start
  select    -- must be conditional to work in INTERRUPT_HANDLER
      Mouse_Buffer.Save.Reticle_Motion;
      --$TP(0058) Mouse rendezvous with Save end
    else
      raise MOUSE_ERROR;
    end select;
  end if;
else
  if BUTTON_PUSHED(Mouse_Data.reset) and
     not PREV_BUTTON_PUSH(Mouse_Data.reset) then
    for I in Status.RESET_STATUS_TYPE loop
      Status.STATUS_CONTROL(I).DATA := 0;
      Status.STATUS_CONTROL(I).DISPLAYED := FALSE;
    end loop;
    Status.REQ_COUNT := Status.REQ_COUNT + 1;
    CHANGE_REQUESTED := TRUE;
    PREV_BUTTON_PUSH(Mouse_Data.reset) := TRUE;
  else
    if not BUTTON_PUSHED(Mouse_Data.reset) then
      PREV_BUTTON_PUSH(Mouse_Data.reset) := FALSE;
    end if;
  end if;
  if BUTTON_PUSHED(Mouse_Data.mode) and
     not PREV_BUTTON_PUSH(Mouse_Data.mode) then
    if Status.MODE = Status.MANUAL then
      Status.MODE := Status.AUTOMATIC;
    else
      Status.MODE := Status.MANUAL;
    end if;
    Status.MODE_DISPLAYED := FALSE;
    Status.REQ_COUNT := Status.REQ_COUNT + 1;
    CHANGE_REQUESTED := TRUE;
    PREV_BUTTON_PUSH(Mouse_Data.mode) := TRUE;
  else
    if not BUTTON_PUSHED(Mouse_Data.mode) then
      PREV_BUTTON_PUSH(Mouse_Data.mode) := FALSE;
    end if;
  end if;
  if BUTTON_PUSHED(Mouse_Data.launch) and
     not PREV_BUTTON_PUSH(Mouse_Data.launch) then
    if Status.MODE = Status.MANUAL then
      Mouse_Buffer.LAUNCH := TRUE;
      Mouse_Buffer.NEW_ABS_X := Mouse_Buffer.OLD_ABS_X;
      Mouse_Buffer.NEW_ABS_Y := Mouse_Buffer.OLD_ABS_Y;
    end if;
```

```
            PREV_BUTTON_PUSH(Mouse_Data.launch) := TRUE;
          else
            if not BUTTON_PUSHED(Mouse_Data.launch) then
              PREV_BUTTON_PUSH(Mouse_Data.launch) := FALSE;
            end if;
          end if;
          if CHANGE_REQUESTED and then Status.REQ_COUNT = 1 then
            --$TP(0059) Mouse rendezvous with Status start
            select
              Status.Update.Signal;
              --$TP(0060) Mouse rendezvous with Status end
            else
              raise MOUSE_ERROR;
            end select;
          end if;
        end if;
        CHANGE_REQUESTED := FALSE;
        REPORT_COUNT := 0;
      when others => null;
    end case;
  Send_Control(Mouse_Data.pic_8259,Mouse_Data.spec_eoi); -- specific EoI
  --$TP(0061) Mouse task end
  end Report;
 end loop;
exception
 when others =>
  Debug_IO.Put_Line("Error in Char_In Task");
  Send_Control(Mouse_Data.pic_8259,Mouse_Data.spec_eoi);
  raise;
end Char_In;

end Mouse;
```

```
-------------------------------------------------------------------------
--| UNIT:     Mouse_Buffer Package Spec.                               --
--| Effects:  Buffers mouse data input, translates it to pixel system.  --
--| Modifies: No global data is modified (other than in own spec).      --
--| Requires: No initialization is required.                           --
--| Raises:   No explicitly raised exceptions are propagated.           --
--| Engineer: M. Sperry.                                               --
-------------------------------------------------------------------------


-- Date    : 10-24-88
-- Purpose :
--   Package Mouse_Buffer contains a task called Save which is responsible for
-- saving reports of mouse movement via a rendezvous with an interrupt task.
-- The task then rendezvous with the display task to relocate the reticle.

with Types;
with Config;

package Mouse_Buffer is

stack_size : constant := 118;  -- in bytes

MOUSE_X     : Types.WORD;     -- for use with the Save task in Mouse_Buffer
MOUSE_Y     : Types.WORD;     -- for use with the Save task in Mouse_Buffer
LAUNCH      : BOOLEAN := FALSE;

OLD_ABS_X  : Types.WORD;     -- absolute X position of Reticle on Screen
OLD_ABS_Y  : Types.WORD;     --      "    Y     "          "         "
NEW_ABS_X  : Types.WORD;     -- for use by ENGAGE (latched values by Mouse pkg)
NEW_ABS_Y  : Types.WORD;

task type Save_Type is
  entry Reticle_Motion;
  pragma PRIORITY(Config.save_priority);
end Save_Type;
for Save_Type'STORAGE_SIZE use INTEGER(Config.bytes_per_storage_unit *
                                                            stack_size);
Save        : Save_Type;     -- for saving motion of mouse to display

end Mouse_Buffer;
```

```
--------------------------------------------------------------------
--| UNIT:     Mouse_Buffer Package Body.                             --
--| Effects:  Buffers mouse data input, translates it to pixel system. --
--| Modifies: No global data is modified (other than in own spec).   --
--| Requires: No initialization is required.                         --
--| Raises:   No explicitly raised exceptions are propagated.        --
--| Engineer: M. Sperry.                                             --
--------------------------------------------------------------------


-- Date     : 10-24-88
-- Purpose :
--    Package body Mouse_Buffer is responsible for the implementation of the
-- buffering between the mouse interrupt routine and the screen.  Note that
-- checks are performed to be sure that the reticle is within the screen
-- defined by Config.  Also, note that the Y coordinate is reversed because
-- the screen on the EGA runs (in the Y direction) from 0 to 349 starting
-- from the upper left and moving down, i.e., the mouse has Y direction as
-- positive moving up, and the EGA has positive moving down.


with Shapes;
with Graphics;
with Config;
with Debug_IO;
with Interrupt_Control;
with Time_Stamp;
pragma ELABORATE(Debug_IO, Graphics, Interrupt_Control, Time_Stamp);

package body Mouse_Buffer is
use Types;                                    -- needed for visibility to '+'


task body Save_Type is

list_len     : constant := 1;

left_limit   : constant := Config.battlefield_screen_left;
right_limit  : constant := Config.battlefield_screen_right;
top_limit    : constant := Config.battlefield_screen_top;
bottom_limit : constant := Config.battlefield_screen_bottom;

PRIORITY     : Graphics.PRIORITY_TYPE := Graphics.HIGH;
WORK_LIST    : Graphics.MOVE_LIST_TYPE(list_len .. list_len);-- 1 item (reticle)

TEMP_X       : Types.WORD;
TEMP_Y       : Types.WORD;

begin
--
-- Initial display of reticle
```

```
--
 WORK_LIST(list_len).XY_OLD := (Config.battlefield_center_x, Config.battlefield_center_y);
 WORK_LIST(list_len).XY_NEW := (Config.battlefield_center_x, Config.battlefield_center_y);
 WORK_LIST(list_len).OBJECT := (Shapes.PIXEL_MODE,Shapes.RETICLE);
 WORK_LIST(list_len).COLOR := Graphics.reticle_color;

 Graphics.Display.Move(PRIORITY, WORK_LIST);

 loop
   begin                   -- exception block
     --$TP(0062) Mouse Buffer task and accept
     accept Reticle_Motion;
--
-- Get new positions of reticle (mouse)
--
     Interrupt_Control.Disable;
     TEMP_X := WORK_LIST(list_len).XY_OLD.X + MOUSE_X;
     TEMP_Y := WORK_LIST(list_len).XY_OLD.Y - MOUSE_Y;
     Interrupt_Control.Enable;
--
--  Check bounds of reticle; don't let it go past edge of screen.
--
     if (TEMP_X + Shapes.RETICLE_LEFT) < left_limit then
       TEMP_X := left_limit - Shapes.RETICLE_LEFT;
     elsif (TEMP_X + Shapes.RETICLE_RIGHT) > right_limit then
       TEMP_X := right_limit - Shapes.RETICLE_RIGHT;
     end if;
     if (TEMP_Y + Shapes.RETICLE_TOP) < top_limit then
       TEMP_Y := top_limit - shapes.reticle_top;
     elsif (TEMP_Y + Shapes.RETICLE_BOTTOM) > bottom_limit then
       TEMP_Y := bottom_limit - Shapes.RETICLE_BOTTOM;
     end if;

     WORK_LIST(list_len).XY_NEW.X := TEMP_X;
     WORK_LIST(list_len).XY_NEW.Y := TEMP_Y;

--
--  update global accessable values
--
     Interrupt_Control.Disable;
     OLD_ABS_X := TEMP_X;
     OLD_ABS_Y := TEMP_Y;
     Interrupt_Control.Enable;

     --$TP(0063) Mouse_Buffer rendezvous with Graphics start
     Graphics.Display.Move(PRIORITY, WORK_LIST);
     --$TP(0064) Mouse_Buffer rendezvous with Graphics end

     WORK_LIST(list_len).XY_OLD := WORK_LIST(list_len).XY_NEW;
   exception
     when others =>
```

```
        Debug_IO.Put_Line("Error in Save");
     end;                                    -- exception block
  --$TP(0065) Mouse_Buffer task end
   end loop;

end Save_Type;

end Mouse_Buffer;
```

```
-------------------------------------------------------------------------
--| UNIT:     RDL Package Body Subunit.                                --
--| Effects:  Supports all Rocket Data Link functions of Simulator.    --
--| Modifies: No global data is modified.                              --
--| Requires: No initialization is required.                           --
--| Raises:   No explicitly raised exceptions are propagated.          --
--| Engineer: T. Griest.                                               --
-------------------------------------------------------------------------


--
--  The RDL package provides tasks to interface to the Rocket Data Link
--  issuing messages for new rocket positions and receiving messages
--  commanding new rocket attitudes.
--
separate(Simulate)
package body RDL is                         --| Rocket Data Link Simulator

  stack_size : constant := 348;

  task type Rock_Sup_Type is
    pragma PRIORITY(Config.rock_sup_priority);
  end Rock_Sup_Type;
  for Rock_Sup_Type'STORAGE_SIZE use INTEGER(Config.bytes_per_storage_unit *
                                              stack_size);
  Rock_Sup : Rock_Sup_Type;

  task body Rock_Sup_Type is separate;

  task body Report_Buf_Type is separate;

  task body Guide_Buf_Type is separate;

end RDL;
```

```
---------------------------------------------------------------------
--| UNIT:     Report_Buf Task Body Subunit.                          --
--| Effects:  Buffers Rocket report data between simulator and Control. --
--| Modifies: No global data is modified.                            --
--| Requires: No initialization is required.                         --
--| Raises:   No explicitly raised exceptions are propagated.        --
--| Engineer: T. Griest.                                             --
---------------------------------------------------------------------


--
--  The Report_Buf task acts as a buffer between the rocket data link
--  support task Rock_Sup and the Rocket.Control task which processes
--  the rocket data.
--
--  TIMING CONSIDERATIONS:  Report_Buf will only provide the most recent
--  message received.  If two messages are received prior to one being
--  taken, the first will be lost.
--
with Debug_IO;
with Time_Stamp;
pragma ELABORATE(Debug_IO, Time_Stamp);

separate (Simulate.RDL)

task body Report_Buf_Type is
  use Types;
  ROCKET_MSG    : Rocket.ROCKET_MSG_TYPE;
begin
  ROCKET_MSG.NUM_ROCKETS := 0;             -- default
  loop
    select
      accept Put_Report(DATA : in Rocket.ROCKET_MSG_TYPE) do
        --$TP(0066) Report_Buf accept Put_Report start
        ROCKET_MSG.NUM_ROCKETS := DATA.NUM_ROCKETS;          -- copy data
        for I in Types.WORD_INDEX range 1..DATA.NUM_ROCKETS loop
          ROCKET_MSG.ROCKET_LIST(I) := DATA.ROCKET_LIST(I);
        end loop;
        --$TP(0067) Report_Buf accept Put_Report end
      end Put_Report;
    or
      accept Get_Report(DATA : out Rocket.ROCKET_MSG_TYPE) do
        --$TP(0068) Report_Buf accept Get_Report start
        DATA.NUM_ROCKETS := ROCKET_MSG.NUM_ROCKETS;
        for I in Types.WORD_INDEX range 1..ROCKET_MSG.NUM_ROCKETS loop
          DATA.ROCKET_LIST(I) := ROCKET_MSG.ROCKET_LIST(I);
        end loop;
        --$TP(0069) Report_Buf accept Get_Report end
      end Get_Report;
    end select;
  end loop;
```

```
exception
  when others =>
    Debug_IO.Put_Line("REPORT_BUF termination due to exception.");
end Report_Buf_Type;
```

```
-----------------------------------------------------------------
--| UNIT:     Rocket Package Spec.                              --
--| Effects:  Provides structure for Rocket managment within BDS.   --
--| Modifies: No global data is modified.                       --
--| Requires: No initialization is required.                    --
--| Raises:   No explicitly raised exceptions are propagated.   --
--| Engineer: T. Griest.                                        --
-----------------------------------------------------------------


--
--  The Rocket package provides all processing to maintain the rockets
--  in flight.
--
with Types;
with Config;

package Rocket is

stack_size              : constant := 1936;      -- in bytes

------------------------------------------
--        REPORT INFORMATION            --
------------------------------------------
  type ROCKET_ITEM_TYPE is record       --| provides essentials on a rocket
      TIME_TAG          : Types.WORD;
      ROCKET_ID         : Types.WORD_INDEX;
      POSITION          : Types.POSITION_TYPE;
    end record;

  type ROCKET_LIST_TYPE is        --| list of all rocket data
    array(Types.WORD_INDEX range <>) of ROCKET_ITEM_TYPE;

  type ROCKET_MSG_TYPE is record
      NUM_ROCKETS       : Types.WORD_INDEX;
      ROCKET_LIST       : ROCKET_LIST_TYPE(1..Config.max_rockets);
  end record;

------------------------------------------
--        GUIDANCE INFORMATION          --
------------------------------------------
  type ROCKET_GUIDE_TYPE is record
    ROCKET_ID           : Types.WORD_INDEX;
    AIMPOINT            : Types.AIMPOINT_TYPE;
  end record;

  type ROCKET_GUIDE_LIST_TYPE is     --| list of all guidance data
    array(Types.WORD_INDEX range <>) of ROCKET_GUIDE_TYPE;

  type ROCKET_GUIDE_MSG_TYPE is record
    NUM_ROCKETS         : Types.WORD_INDEX;
    ROCKET_GUIDE_LIST     : ROCKET_GUIDE_LIST_TYPE(1..Config.max_rockets);
  end record;

-------------------------------------------------------------------------------
  task type Control_Type is                  --| for overall engagement control
    entry Get_Next_Report(ROCKET_REPORT_MSG : in ROCKET_MSG_TYPE);
    pragma PRIORITY(Config.control_priority);
  end Control_Type;
  for Control_Type'STORAGE_SIZE use INTEGER(Config.bytes_per_storage_unit *
                                                        stack_size);
  Control               : Control_Type;

end Rocket;   -- package specification
```

```
----------------------------------------------------------------------
--| UNIT:     Rocket Package Body.                                   --
--| Effects:  Provides structure for Rocket managment within BDS.    --
--| Modifies: No global data is modified.                            --
--| Requires: No initialization is required.                         --
--| Raises:   No explicitly raised exceptions are propagated.        --
--| Engineer: T. Griest.                                             --
----------------------------------------------------------------------


--
--  The Rocket package provides all processing to maintain the rockets
--  in flight.
--
with Debug_IO;
with Status;      -- maintains number Rockets Active
with Shapes;                                   -- for rocket shapes
with Graphics;    -- for graphics operations/colors
with Distrib;
pragma ELABORATE(Debug_IO, Status, Graphics);

package body Rocket is

guidance_stack_size          : constant := 660;

--
--  The rocket guidance activity is given overall control by the Control task.
--  "Control" is used to accept rocket reports, and is responsible for engaging
--  the targets, providing updates to the Graphics.Display task, and generating
--  the guidance messages for the Rocket Data Link.  It achieves much of this
--  with the assistance of one (or more) Guidance task(s).  The Guidance task
--  is responsible for taking a set of the rockets and producing a new
--  aimpoint for each rocket/target in that set.  The activities of the
--  guidance task(s), as well as the Control task can be overlapped
--  considerably, and therefore may benefit from the addition of processors.
--

  GUIDANCE_LIST_ERROR  : exception; -- if guidance list does not match history
--
--  This history data is provided to a guidance task, which in turn processes
--  it and returns the next guidance information needed for each rocket.
--
  type HISTORY_DATA_TYPE is record          --| containing rocket information
    ACTIVE               : BOOLEAN;         --| if rocket was previously active
    TARGET               : Types.WORD_INDEX;  --| rocket's target
    POSITION_PAIR        : Types.POSITION_PAIR_TYPE;
  end record;

  type HISTORY_LIST_TYPE is
                    array(Types.WORD_INDEX range <>) of HISTORY_DATA_TYPE;
```

```
type AIMPOINT_LIST_TYPE is
                      array(Types.WORD_INDEX range <>) of Types.AIMPOINT_TYPE;

ROCKET_HISTORY    : HISTORY_LIST_TYPE(1..Config.max_rockets);

NEXT_GUIDE_MSG    : ROCKET_GUIDE_MSG_TYPE;

 task type Guidance_Type is
   entry History(HISTORY_DATA : in HISTORY_LIST_TYPE);
   entry Next_Guidance(AIMPOINT_LIST : out AIMPOINT_LIST_TYPE);
   pragma PRIORITY(Config.guidance_priority);
 end Guidance_Type;
 for Guidance_Type'STORAGE_SIZE use INTEGER(Config.bytes_per_storage_unit *
                                           guidance_stack_size);

 Rocket_Guide : array(Types.WORD_INDEX range 1..Distrib.num_guide_tasks)
                                                        of Guidance_Type;

  task body Guidance_Type is separate;

  task body Control_Type is separate;

end Rocket;  -- package body
```

```
-----------------------------------------------------------------------
--| UNIT:     Rock_Sup Task Body Subunit.                            --
--| Effects:  Provides all Rocket Support for Simulator, including    --
--|           target intercept detection.                           --
--| Modifies: Updates state of rockets and targets in Simulator DBase. --
--| Requires: No initialization is required.                         --
--| Raises:   No explicitly raised exceptions are propagated.         --
--| Engineer: T. Griest.                                             --
-----------------------------------------------------------------------


-----------------------------------------------------------------------------
--              ROCK_SUP   Task   Body                                    --
-----------------------------------------------------------------------------
--  The rocket support task provides the necessary rocket motion, based
--  on previous position and the application of a new guidance aimpoint.
--  It generates a new report "ROCKET_MSG" for a buffer task (Report_Buf)
--  to forward to the GDS Rocket.Control task.  Likewise, the Rocket.Control
--  task issues guidance messages to the buffer task (Guide_Buf) which are
--  made available to the Rock_Sup task.  ROCKET/TARGET intercepts are
--  checked in the shared data base within the simulator.  In such cases,
--  both the rocket and target are destroyed (marked inactive).
--
-- Copyright(C) 1988, LabTek Corporation.  Permission is granted to copy
-- and/or use this software provided that this copyright notice is included
-- and all liability for its use is accepted by the user.
--
with Traject;           -- trajectory planner
with Calendar;
with Interrupt_Control;
with Time_Stamp;
pragma ELABORATE(Traject, Calendar, Interrupt_Control, Time_Stamp);

separate (Simulate.RDL)

task body Rock_Sup_Type is

  use Calendar;          -- for - operator
  use Types;             -- for operators
  start_position        : constant Types.POSITION_TYPE :=
                                  (Config.meters_in_battle_area/2.0,60.0,0.0);
  ROCKET_MSG            : Rocket.ROCKET_MSG_TYPE;
  GUIDE_MSG             : Rocket.ROCKET_GUIDE_MSG_TYPE;
  GUIDE_MSG_INDEX       : Types.WORD_INDEX;
  REPORT_MSG_INDEX      : Types.WORD_INDEX;
  POSITION              : Types.POSITION_TYPE;  -- temp
  TIME_TAG              : Types.WORD := 0;
  START_TIME            : Calendar.TIME;
  DELAY_PERIOD          : DURATION;
--
--  MAKE_REPORT: process current rocket ID
```

```
--
 procedure Make_Report(ID : Types.WORD_INDEX; POS : Types.POSITION_TYPE) is
                              --| checks if rocket has collided with
                              --| any targets or ground.  If so, delete
                              --| target(s) and rocket.
   DELTA_X            : Types.METERS;
   DELTA_Y            : Types.METERS;
   DELTA_Z            : Types.METERS;
   DELTA_T            : Types.LONG_FIXED;  -- time for rocket to reach ground
   ROCKET_POS         : Types.POSITION_TYPE;
 begin   -- of  Make_Report
   --$TP(0070) Rock_Sup.Make_Report start
   if POS.Z < 0.0 then
     ROCKETS(ID).ACTIVE := FALSE;           -- destroy rocket
--
-- compute time it took to get to zero
--
     DELTA_X := POS.X - ROCKETS(ID).POSITION.X;
     DELTA_Y := POS.Y - ROCKETS(ID).POSITION.Y;
     DELTA_Z := POS.Z - ROCKETS(ID).POSITION.Z;

     if DELTA_Z = 0.0 then
       DELTA_T := 0.0;
     else
       DELTA_T := Types.LONG_FIXED(ROCKETS(ID).POSITION.Z/abs(DELTA_Z));
     end if;
--
-- find terminal position of Rocket
--
     ROCKET_POS.X := ROCKETS(ID).POSITION.X + Types.METERS(DELTA_T*DELTA_X);
     ROCKET_POS.Y := ROCKETS(ID).POSITION.Y + Types.METERS(DELTA_T*DELTA_Y);
--TBD since targets are always at Z=0, collision point is always 0
--     ROCKET_POS.Z := ROCKETS(ID).POSITION.Z + Types.METERS(DELTA_T*DELTA_Z);
--
--  Now search target list to see if any targets within "kill_radius"
--  perimeter of rocket
--
     for TARGET_ID in TARGETS'range loop
       Interrupt_Control.Disable;                -- access to shared data
       if TARGETS(TARGET_ID).ACTIVE then
         DELTA_X := ROCKET_POS.X - TARGETS(TARGET_ID).POSITION.X;
         DELTA_Y := ROCKET_POS.Y - TARGETS(TARGET_ID).POSITION.Y;
--TBD should use distance DISTANCE := Math.Sqrt( Types.METERS(DELTA_X*DELTA_X) +
--TBD                               Types.METERS(DELTA_Y*DELTA_Y) +
--TBD                               Types.METERS(DELTA_Z*DELTA_Z)));
         if abs DELTA_X < Config.kill_radius and  -- this makes square box
            abs DELTA_Y < Config.kill_radius      -- around each target
         then
           TARGETS(TARGET_ID).ACTIVE := FALSE;  -- destroy target
         end if;
       end if;
```

```
        Interrupt_Control.Enable;
      end loop;
    else                        -- Rocket did not hit ground or target
      REPORT_MSG_INDEX := REPORT_MSG_INDEX + 1;
      ROCKET_MSG.ROCKET_LIST(REPORT_MSG_INDEX) := (TIME_TAG,ID,POS);
    end if;
    --$TP(0071) Rock_Sup.Make_Report end
  end Make_Report;

----------------------------------------------------
--        ROCKET SUPPORT TASK BODY                --
----------------------------------------------------
begin
  for ID in ROCKETS'range loop                -- initialize to all inactive
    ROCKETS(ID).ACTIVE := FALSE;
  end loop;
  START_TIME := Calendar.CLOCK;               -- find out when xeq begins

  loop
    --$TP(0072) Rock_Sup task start
    START_TIME := START_TIME + Config.interval;
    if TIME_TAG = Types.WORD'last then        -- update TIME_TAG to be able
      TIME_TAG := 0;                          --  to differentiate between
    else                                      --  stale and new reports
      TIME_TAG := TIME_TAG + 1;
    end if;

    --$TP(0073) Rock_Sup rendezvous with Guide_Buf start
    RDL.Guide_Buf.Get_Guide(GUIDE_MSG);       -- fetch latest guidance message
    --$TP(0074) Rock_Sup rendezvous with Guide_Buf end
--
--  Go through each rocket, and if active, apply trajectory to
--  current position for 1 interval.
--
    GUIDE_MSG_INDEX := 1;                    -- pointer msg.rocket_guide_list
    REPORT_MSG_INDEX := 0;
    for ROCKET_ID in ROCKETS'range loop
      if GUIDE_MSG_INDEX <= GUIDE_MSG.NUM_ROCKETS and then
         ROCKET_ID = GUIDE_MSG.ROCKET_GUIDE_LIST(GUIDE_MSG_INDEX).ROCKET_ID
      then
--
--   This rocket is in the list, see if it was previously active
--
        if not ROCKETS(ROCKET_ID).ACTIVE then
--
--  filter out guidance messages for rockets that have recently been
--  destroyed (but BDS doesn't know it yet)
--
          if GUIDE_MSG.ROCKET_GUIDE_LIST(GUIDE_MSG_INDEX).AIMPOINT.ELEVATION = 16384
          then        -- a new launch
            ROCKETS(ROCKET_ID).ACTIVE := TRUE; -- launch
          ROCKETS(ROCKET_ID).POSITION := start_position;
          Make_Report(ROCKET_ID,start_position);  -- start at launcher
        end if;
      else
--
--   Now compute new X,Y,Z position.
--
        POSITION := Traject( ROCKETS(ROCKET_ID).POSITION,
                       GUIDE_MSG.ROCKET_GUIDE_LIST(GUIDE_MSG_INDEX).AIMPOINT);
        Make_Report(ROCKET_ID,POSITION);
        ROCKETS(ROCKET_ID).POSITION := POSITION;
      end if;    -- rocket active check
      GUIDE_MSG_INDEX := GUIDE_MSG_INDEX + 1;
    else                                     -- no guidance for this rocket
      if ROCKETS(ROCKET_ID).ACTIVE then
--
--  no guidance information for active rocket, simply don't move it
--
        POSITION := ROCKETS(ROCKET_ID).POSITION;
        Make_Report(ROCKET_ID,POSITION);
      end if;       -- rocket active check
    end if;         -- guide entry exists check
  end loop;
--
--  New report list has been generated.  Send it to buffer task.
--
  ROCKET_MSG.NUM_ROCKETS := REPORT_MSG_INDEX;
  --$TP(0075) Rock_Sup rendezvous with Report_Buf start
  RDL.Report_Buf.Put_Report(ROCKET_MSG);    -- issue next rocket report
  --$TP(0076) Rock_Sup rendezvous with Report_Buf end
--
--  Delay to make rocket motion reports periodic
--
  DELAY_PERIOD := START_TIME - Calendar.CLOCK;
  if DELAY_PERIOD < 0.0 then
    START_TIME := CLOCK;
  end if;
  --$TP(0077) Rock_Sup task end
  delay DELAY_PERIOD;
 end loop;
end Rock_Sup_Type;
```

```
-----------------------------------------------------------------
--| UNIT:     Shapes Package Spec.                             --
--| Effects:  Provides all graphics symbology.                 --
--| Modifies: No global data is modified.                      --
--| Requires: No initialization is required.                   --
--| Raises:   No explicitly raised exceptions are propagated.  --
--| Engineer: T. Griest / M. Sperry.                           --
-----------------------------------------------------------------


-------------------------------------------------
--       SHAPES PACKAGE SPECIFICATION          --
-------------------------------------------------
-- Date    : 10-12-88
-- Purpose :
--   Package Shapes is responsible for determining the shapes of the various
-- symbols.

with Types;
with Config;

package Shapes is

type SYMBOL_TYPE is (ROCKET, TARGET, RETICLE, DOT, ZERO, ONE, TWO, THREE, FOUR,
                     FIVE, SIX, SEVEN, EIGHT, NINE, HORIZONTAL, VERTICAL);

type OBJECT_MODE_TYPE is (TEXT_MODE, PIXEL_MODE);

type PIXEL is record
 X:Types.COORDINATE range Config.entire_screen_left..Config.entire_screen_right;
 Y:Types.COORDINATE range Config.entire_screen_top..Config.entire_screen_bottom;
end record;

type REL_PIXEL is record                        -- offset from base of pixel
  X_OFFSET : Types.REL_COORDINATE;                   -- positive goes right
  Y_OFFSET : Types.REL_COORDINATE;                   -- positive goes down
end record;

type PIXEL_LIST is array(Types.WORD_INDEX range <>) of REL_PIXEL;

type OBJECT_TYPE(OBJECT_MODE : OBJECT_MODE_TYPE := TEXT_MODE) is record
  case OBJECT_MODE is
    when TEXT_MODE =>
      TEXT_OBJECT  : STRING(1..Config.stats_title_max_length);
    when PIXEL_MODE =>
      PIXEL_OBJECT : SYMBOL_TYPE;
  end case;
end record;

type OBJECT_PTR is access PIXEL_LIST;
```

```
reticle_left   : constant := -5;              -- constants used to check if
reticle_right  : constant :=  5;              -- reticle going past screen
reticle_top    : constant := -5;              -- boundaries.
reticle_bottom : constant :=  5;
--
--  The following two constants determine how far the target center can
--  be in meters from the indicated reticle center and still allow
--  aquisition of the target for launching a rocket.  They are not the
--  same in X and Y, since the reticle is slightly rectangular.
--
reticle_x_error: constant := 40.25;     -- METERS to allow target aquisition
reticle_y_error: constant := 49.50;     -- METERS to allow target aquisition

NUMERIC         : array(0..9) of SYMBOL_TYPE := (ZERO, ONE, TWO, THREE, FOUR,
                                          FIVE, SIX, SEVEN, EIGHT, NINE);

number_width   : constant := 8;               -- widest number in pixels
OBJECT_PTR_TABLE : array(SYMBOL_TYPE) of OBJECT_PTR :=
  (TARGET => new PIXEL_LIST'(
                              (0,-2),
                           (-1,-1), (1,-1),
                        (-2,0), (0,0), (2,0),
                           (-1,1), (1,1),
                              (0,2) ),

   ROCKET => new PIXEL_LIST'(
                           (0,0),
                           (0,1),
                           (0,2),
                           (0,3),
                     (-1,4),    (1,4)  ),

   RETICLE => new PIXEL_LIST'(
(-5,-5),(-4,-5),(-3,-5),                              (3,-5),(4,-5),(5,-5),
(-5,-4),                                                            (5,-4),
(-5,-3),                        (0,-3),                             (5,-3),
                                (0,-2),
                                (0,-1),
          (-3,0), (-2,0), (-1,0), (0,0),  (1,0),  (2,0),  (3,0),
                                (0,1),
                                (0,2),
(-5,3),                         (0,3),                              (5,3),
(-5,4),                                                             (5,4),
(-5,5),(-4,5),(-3,5),                                 (3,5),(4,5),(5,5)),

   DOT => new PIXEL_LIST'( (0,0),(0,0) ),

   ZERO => new PIXEL_LIST'(        (1,-8),(2,-8),(3,-8),(4,-8),(5,-8),
                           (0,-7),                               (6,-7),
                           (0,-6),                         (5,-6),(6,-6),
                           (0,-5),                   (4,-5),      (6,-5),
                          (0,-4),              (3,-4),               (6,-4),
                          (0,-3),       (2,-3),                      (6,-3),
                          (0,-2),(1,-2),                             (6,-2),
                          (0,-1),                                    (6,-1),
                                 (1,0), (2,0), (3,0), (4,0), (5,0)),

ONE => new PIXEL_LIST'(          (4,-8),
                          (3,-7),(4,-7),
                                 (4,-6),
                                 (4,-5),
                                 (4,-4),
                                 (4,-3),
                                 (4,-2),
                                 (4,-1),
                          (3,0), (4,0), (5,0)),

 TWO => new PIXEL_LIST'(          (1,-8),(2,-8),(3,-8),(4,-8),
                           (0,-7),                              (5,-7),
                                                                (5,-6),
                                                         (4,-5),
                                                  (3,-4),
                                           (2,-3),
                                  (1,-2),
                           (0,-1),
                           (0,0), (1,0), (2,0), (3,0), (4,0), (5,0)),

  THREE => new PIXEL_LIST'(        (1,-8),(2,-8),(3,-8),
                           (0,-7),                      (4,-7),
                                                        (4,-6),
                                                        (4,-5),
                                          (2,-4),(3,-4),
                                                        (4,-3),
                                                        (4,-2),
                            (0,-1),                     (4,-1),
                                   (1,0), (2,0), (3,0)),

  FOUR => new PIXEL_LIST'(                                (4,-8),
                                                   (3,-7),(4,-7),
                                            (2,-6),       (4,-6),
                                     (1,-5),              (4,-5),
                            (0,-4),(1,-4),(2,-4),(3,-4),(4,-4),(5,-4),
                                                          (4,-3),
                                                          (4,-2),
                                                          (4,-1),
                                                   (3,0), (4,0), (5,0)),

   FIVE => new PIXEL_LIST'(         (1,-8),(2,-8),(3,-8),(4,-8),(5,-8),
                            (0,-7),
                            (0,-6),
                            (0,-5),
                                    (1,-4),(2,-4),(3,-4),(4,-4),
                                                      (5,-3),
                                                      (5,-2),
                                                      (5,-1),
                          (0,0),  (1,0), (2,0), (3,0), (4,0), (5,0)),

SIX => new PIXEL_LIST'(                       (3,-8),(4,-8),
                                         (2,-7),
                                   (1,-7),
                          (0,-6),
                          (0,-5), (1,-5),(2,-5),(3,-5),
                          (0,-4),                      (4,-4),
                          (0,-3),                      (4,-3),
                          (0,-2),                      (4,-2),
                          (0,-1),                      (4,-1),
                                  (1,0), (2,0), (3,0)),

SEVEN => new PIXEL_LIST'(         (1,-8),(2,-8),(3,-8),(4,-8),(5,-8),
                          (0,-7),                             (5,-7),
                                                       (4,-6),
                                                (3,-5),
                                         (2,-4),
                                  (1,-3),
                          (0,-2),
                          (0,-1),
                          (0,0)),

EIGHT => new PIXEL_LIST'(         (1,-8),(2,-8),(3,-8),(4,-8),
                          (0,-7),                             (5,-7),
                          (0,-6),                             (5,-6),
                          (0,-5),                             (5,-5),
                                  (1,-4),(2,-4),(3,-4),(4,-4),
                          (0,-3),                             (5,-3),
                          (0,-2),                             (5,-2),
                          (0,-1),                             (5,-1),
                                  (1,0), (2,0), (3,0), (4,0)),

NINE => new PIXEL_LIST'(          (1,-8),(2,-8),(3,-8),(4,-8),
                          (0,-7),                             (5,-7),
                          (0,-6),                             (5,-6),
                          (0,-5),                             (5,-5),
                                  (1,-4),(2,-4),(3,-4),(4,-4),(5,-4),
                                                              (5,-3),
                                                       (4,-2),
                                                (3,-1),
                                  (1,0), (2,0)),
HORIZONTAL => new PIXEL_LIST'((0, 0),(1, 0),(2, 0),(3, 0),(4, 0),(5, 0),
                              (6, 0),(7, 0),(8, 0),(9, 0),(10,0),(11,0),
                              (12,0),(13,0),(14,0),(15,0),(16,0),(17,0),
                              (18,0),(19,0),(20,0),(21,0),(22,0),(23,0),
                              (24,0),(25,0),(26,0),(27,0),(28,0),(29,0),
                              (30,0),(31,0),(32,0),(33,0),(34,0),(35,0),
                                        (36,0),(37,0),(38,0),(39,0),(40,0),(41,0),
                                        (42,0),(43,0),(44,0),(45,0),(46,0),(47,0),
                                        (48,0),(49,0),(50,0),(51,0),(52,0),(53,0),
                                        (54,0),(55,0),(56,0),(57,0),(58,0),(59,0),
                                        (60,0),(61,0),(62,0),(63,0),(64,0),(65,0),
                                        (66,0),(67,0),(68,0),(69,0),(70,0),(71,0),
                                        (72,0),(73,0),(74,0),(75,0),(76,0),(77,0),
                                        (78,0)),
   VERTICAL =>  new PIXEL_LIST'((0, 0),(0, 1),(0, 2),(0, 3),(0, 4),(0, 5),
                                        (0, 6),(0, 7),(0, 8),(0, 9),(0,10),(0,11),
                                        (0,12),(0,13),(0,14),(0,15)));

end Shapes;
```

```
--------------------------------------------------------------------------
--| UNIT:     Sensor Package Body Subunit.                              --
--| Effects:  Provides structure for all simulator Target motion.       --
--| Modifies: Simulator target data is updated.                         --
--| Requires: No initialization is required.                            --
--| Raises:   No explicitly raised exceptions are propagated.           --
--| Engineer: T. Griest.                                                --
--------------------------------------------------------------------------


--
--  Simulator package to provide testing of BDS system
--
separate(Simulate)

package body Sensor is                   --| Target Sensor Simulator
  task body Targ_Sup_Type is separate;
end Sensor;          -- body
```

```
-------------------------------------------------------------------------
--| UNIT:     Simulate Package Spec.                                   --
--| Effects:  Provides shared data base for Simulator.                 --
--| Modifies: No global data is modified.                              --
--| Requires: Individual tasks are responsible for init. of global data.--
--| Raises:   No explicitly raised exceptions are propagated.          --
--| Engineer: T. Griest.                                               --
-------------------------------------------------------------------------


--
--  Simulator package to provide testing of BDS system
--
with Target;
with Rocket;
with Sync;
with Config;

package Simulate is                   --| Overall simulation package

  package Sensor is                   --| Target Sensor Simulator
    stack_size : constant := 114;     -- in bytes
    task type Targ_Sup_Type is
      pragma PRIORITY(Config.targ_sup_priority);
      entry Next_Target_Msg(Data : out Target.TARGET_MSG_TYPE);
      entry Clock(Time : in  Sync.TIME_TYPE);
    end Targ_Sup_Type;
    for Targ_Sup_Type'STORAGE_SIZE use INTEGER(Config.bytes_per_storage_unit *
                                                              stack_size);
    Targ_Sup : Targ_Sup_Type;
  end Sensor;

  package RDL is                      --| Rocket Data Link Simulator

    report_buf_stack_size : constant := 302;      -- in bytes
    guide_buf_stack_size  : constant := 744;      -- in bytes

--
-- The Report_Buf task buffers Rocket Reports from the Rock_Sup task
-- and provides them to the Rocket.Control task
--
    task type Report_Buf_Type is
      pragma PRIORITY(Config.report_buf_priority);
      entry Put_Report(DATA : in  Rocket.ROCKET_MSG_TYPE);
      entry Get_Report(DATA : out Rocket.ROCKET_MSG_TYPE);
    end Report_Buf_Type;
    for Report_Buf_Type'STORAGE_SIZE use INTEGER(Config.bytes_per_storage_unit
                                            * report_buf_stack_size);
    Report_Buf : Report_Buf_Type;

--
```

```
-- The Guide_Buf task buffers new Guidance messages from the Rocket.Control
-- task for delivery to the Rock_Sup task.
--
    task type Guide_Buf_Type is
      pragma PRIORITY(Config.guide_buf_priority);
      entry Put_Guide(DATA : in  Rocket.ROCKET_GUIDE_MSG_TYPE);
      entry Get_Guide(DATA : out Rocket.ROCKET_GUIDE_MSG_TYPE);
    end Guide_Buf_Type;
    for Guide_Buf_Type'STORAGE_SIZE use INTEGER(Config.bytes_per_storage_unit
                                                * guide_buf_stack_size);
    Guide_Buf : Guide_Buf_Type;

  end RDL;

end Simulate;
```

```
----------------------------------------------------------------------
--| UNIT:     Simulate Package Body.                                 --
--| Effects:  Provides shared data base for Simulator.              --
--| Modifies: No global data is modified.                            --
--| Requires: Individual tasks are responsible for init. of global data.--
--| Raises:   No explicitly raised exceptions are propagated.        --
--| Engineer: T. Griest.                                             --
----------------------------------------------------------------------


with Types;

--
--  Simulator package to provide testing of BDS system
--
package body Simulate is                    --| Overall simulation package

----------------------------------------------
--   TARGET DATA                            --
----------------------------------------------
  type TARGET_SIM_TYPE is record      --| provides individual target information
    ACTIVE           : BOOLEAN;
    POSITION         : Types.POSITION_TYPE;
    TARGET_CLASS     : Types.TARGET_CLASS_TYPE;
  end record;

  type TARGETS_TYPE is
        array(Types.WORD_INDEX range 1..Config.max_targets) of TARGET_SIM_TYPE;

  TARGETS : TARGETS_TYPE;

----------------------------------------------
--   ROCKET DATA                            --
----------------------------------------------
  type ROCKET_SIM_TYPE is record      --| provides individual rocket information
    ACTIVE           : BOOLEAN;
    POSITION         : Types.POSITION_TYPE;
  end record;

  type ROCKETS_TYPE is
       array(Types.WORD_INDEX range 1..Config.max_rockets) of ROCKET_SIM_TYPE;

  ROCKETS : ROCKETS_TYPE;

  package body Sensor is separate;          --| Target Sensor Simulator

  package body RDL is separate;             --| Rocket Data Link Simulator

end Simulate;               -- body
```

```
--------------------------------------------------------------------
--| UNIT:     Status Package Spec.                                --
--| Effects:  Maintains indicators and statistics on graphics display.  --
--| Modifies: Flags are cleared in spec. when values are displayed.    --
--| Requires: Initialization must be signaled by main for first display.--
--| Raises:   No explicitly raised exceptions are propagated.          --
--| Engineer: M. Sperry.                                              --
--------------------------------------------------------------------


-- Date     : 11-08-88
-- Purpose :
--   The purpose of the Status specification package is to provide visibility
-- to the data base which holds the requests from the mouse, et. al.  The
-- requests are entered into a data table (called STATUS_CONTROL) and then
-- the table is checked to see if any updating of the statistics needs to be
-- done.  The checking of the table is done at an atomic level to prevent
-- the shared data from being corrupted at critical times. The commands are
-- processed from the mouse interrupt as mode first, then reset if there are
-- two commands to perform.

with Types;
with Config;

package Status is

stack_size       : constant := 252;

type MODE_TYPE is (AUTOMATIC,MANUAL);

type STATUS_TYPE is (AIRBORNE, TRACKED, EXPENDED, DESTROYED);

subtype RESET_STATUS_TYPE is STATUS_TYPE range EXPENDED..DESTROYED;

type STATUS_RECORD is record
  DATA      : Types.WORD := 0;                  -- new statistic
  DISPLAYED : BOOLEAN := FALSE;                 -- need to display
end record;


type STATUS_TYPE_ARRAY is array(STATUS_TYPE'FIRST .. STATUS_TYPE'LAST) of
                                                        STATUS_RECORD;
--
-- define shared variables
--

MODE             : MODE_TYPE := MANUAL;
MODE_DISPLAYED   : BOOLEAN := FALSE;
STATUS_CONTROL   : STATUS_TYPE_ARRAY;
REQ_COUNT        : Types.WORD := 0;
```

```
STATUS_ERROR     : EXCEPTION;                    -- if data negative

--
-- define subprograms and tasks
--

procedure Initialize;                            -- initialization of screen

task type Update_Type is
  entry Signal;
  pragma PRIORITY(Config.update_priority);
end Update_Type;
for Update_Type'STORAGE_SIZE use INTEGER(Config.bytes_per_storage_unit *
                                                 stack_size);
Update           : Update_Type;

end Status;
```

```
-------------------------------------------------------------------
--| UNIT:     Status Package Body.                                --
--| Effects:  Maintains indicators and statistics on graphics display.  --
--| Modifies: Flags are cleared in spec. when values are displayed.     --
--| Requires: Initialization must be signaled for first display.        --
--| Raises:   No explicitly raised exceptions are propagated.           --
--| Engineer: M. Sperry.                                                --
-------------------------------------------------------------------


-- Date    : 11-08-88
-- Purpose :
--   The purpose of the status package body is the implementation of the status
-- update task.  Although operating at a low priority, the update task updates
-- the various statistics by a rendezvous with the graphics task.

with Graphics;
with Interrupt_Control;
with Shapes;
with Machine_Dependent;
with Interrupt_Control;
with Debug_IO;
with Time_Stamp;
pragma ELABORATE(Graphics, Interrupt_Control, Debug_IO, Time_Stamp);

package body Status is
use Types;                                    -- for visibility to '+';

procedure Initialize is
-- A procedure which initializes the screen for the various statistics
-- descriptions.  It also signals the status update task.

TITLE_PRIORITY : Graphics.PRIORITY_TYPE := Graphics.LOW;

TITLES : Graphics.MOVE_LIST_TYPE(1..12) :=
        (((0,0),(0,0),(Shapes.TEXT_MODE,"Airborne   "),Graphics.status_color),
         ((0,0),(0,1),(Shapes.TEXT_MODE,"  Rockets: "),Graphics.status_color),
         ((0,0),(0,3),(Shapes.TEXT_MODE,"Tracked    "),Graphics.status_color),
         ((0,0),(0,4),(Shapes.TEXT_MODE,"  Targets: "),Graphics.status_color),
         ((0,0),(0,8),(Shapes.TEXT_MODE,"Totals     "),Graphics.status_color),
        ((0,0),(0,10),(Shapes.TEXT_MODE,"Expended   "),Graphics.status_color),
        ((0,0),(0,11),(Shapes.TEXT_MODE,"  Rockets: "),Graphics.status_color),
        ((0,0),(0,13),(Shapes.TEXT_MODE,"Destroyed  "),Graphics.status_color),
        ((0,0),(0,14),(Shapes.TEXT_MODE,"  Targets: "),Graphics.status_color),
        ((0,0),(0,18),(Shapes.TEXT_MODE,"Mode:      "),Graphics.status_color),
        ((0,0),(0,20),(Shapes.TEXT_MODE,"  Manual   "),Graphics.status_color),
        ((0,0),(0,22),(Shapes.TEXT_MODE,"Automatic  "),Graphics.status_color));

begin
  Graphics.Display.Move(TITLE_PRIORITY,TITLES);
  Interrupt_Control.Disable;                 -- go atomic
  Status.REQ_COUNT := Status.REQ_COUNT + 1;   -- signal a request (print zeroes)
  Interrupt_Control.Enable;
  Status.Update.Signal;                       -- display statistics values
end Initialize;

task body Update_Type is
use Types;                                      -- for visibility to "+"

x_start            : constant := 11;            -- column that status_box starts in x
x_end              : constant := 90;            -- end column of status_box
y_top_start_A      : constant := 307;           -- status_box top AUTOMATIC
y_bottom_start_A   : constant := 322;           -- status_box bottom AUTOMATIC
y_top_start_M      : constant := 278;           -- status_box top MANUAL
y_bottom_start_M   : constant := 293;           -- status_box bottom MANUAL
manual_offset      : constant := 29;            -- offset to draw status_box
box_start          : constant := 1;             -- range of components that
box_end            : constant := 4;             -- make up status_box.
base_x             : constant Types.COORDINATE := 120;-- x end of all statistics
airborne_y         : constant Types.COORDINATE := 25;  -- y location of stat
tracked_y          : constant Types.COORDINATE := 67;  -- y location of stat
expended_y         : constant Types.COORDINATE := 165; -- y location of stat
destroyed_y        : constant Types.COORDINATE := 207; -- y location of stat

y_statistics       : constant array(STATUS_TYPE'first .. STATUS_TYPE'last) of
      Types.COORDINATE := (airborne_y, tracked_y, expended_y, destroyed_y);

type STATUS_OLD is array(STATUS_TYPE'first .. STATUS_TYPE'last,
                    1 .. Config.statistics_length) of Graphics.MOVE_RECORD;

NEXT_MODE          : MODE_TYPE;
DISPLAY_REQUIRED   : BOOLEAN;
NEXT_DATA          : Types.WORD;
BOX_LIST           : Graphics.MOVE_LIST_TYPE(Types.WORD_INDEX range box_start..box_end);
DATA_OLD           : STATUS_OLD;
WORK_LIST          : Graphics.MOVE_LIST_TYPE(1 .. Config.statistics_length);
MOVE_PRIORITY      : Graphics.PRIORITY_TYPE := Graphics.LOW;


procedure Initialize is

-- A procedure which intializes the DATA_OLD data base.  This procedure does
-- NOT cause the digits to be drawn.  Then, it initializes the status_box around
-- 'manual'.  Again, it does not cause the status_box to be drawn.  A wakeup
-- from the main task will cause it to be drawn.

begin
  for I in STATUS_TYPE'first .. STATUS_TYPE'last loop
    for J in 1 .. Config.statistics_length loop
      DATA_OLD(I,J).XY_OLD := (Types.COORDINATE(base_x),Types.COORDINATE(y_statistics(I)));
      DATA_OLD(I,J).XY_NEW := (Types.COORDINATE(base_x),Types.COORDINATE(y_statistics(I)));
      DATA_OLD(I,J).OBJECT := (Shapes.PIXEL_MODE,Shapes.ZERO);
      DATA_OLD(I,J).COLOR  := Graphics.status_color;
    end loop;
  end loop;
--
-- Now initialize top of status_box
--
  BOX_LIST(1).XY_OLD :=
            (Types.COORDINATE(x_start),Types.COORDINATE(y_top_start_A));
  BOX_LIST(1).XY_NEW :=
            (Types.COORDINATE(x_start),Types.COORDINATE(y_top_start_A));
  BOX_LIST(1).OBJECT := (Shapes.PIXEL_MODE,Shapes.HORIZONTAL);
  BOX_LIST(1).COLOR  := Graphics.status_box_color;
--
-- define bottom of status_box
--
  BOX_LIST(2).XY_OLD :=
            (Types.COORDINATE(x_start), Types.COORDINATE(y_bottom_start_A));
  BOX_LIST(2).XY_NEW :=
            (Types.COORDINATE(x_start), Types.COORDINATE(y_bottom_start_A));
  BOX_LIST(2).OBJECT := (Shapes.PIXEL_MODE,Shapes.HORIZONTAL);
  BOX_LIST(2).COLOR  := Graphics.status_box_color;
--
-- define left side of status_box
--
  BOX_LIST(3).XY_OLD :=
            (Types.COORDINATE(x_start), Types.COORDINATE(y_top_start_A));
  BOX_LIST(3).XY_NEW :=
            (Types.COORDINATE(x_start), Types.COORDINATE(y_top_start_A));
  BOX_LIST(3).OBJECT := (Shapes.PIXEL_MODE,Shapes.VERTICAL);
  BOX_LIST(3).COLOR  := Graphics.status_box_color;
--
-- define right side of status_box
--
  BOX_LIST(4).XY_OLD :=
            (Types.COORDINATE(x_end), Types.COORDINATE(y_top_start_A));
  BOX_LIST(4).XY_NEW :=
            (Types.COORDINATE(x_end), Types.COORDINATE(y_top_start_A));
  BOX_LIST(4).OBJECT := (Shapes.PIXEL_MODE,Shapes.VERTICAL);
  BOX_LIST(4).COLOR  := Graphics.status_box_color;
exception
  when others => Debug_IO.Put_Line("Exception raised in Status.Initialize");
end Initialize;


procedure Update_Box(NEXT_MODE : MODE_TYPE) is

-- A procedure which updates the four objects which represent the status_box
-- surrounding one of the modes.

OFFSET : Types.WORD;
```

```
begin
  --$TP(0078) Status.Update_Box start
  if NEXT_MODE = AUTOMATIC then            -- draw status_box at 'automatic'
    BOX_LIST(1).XY_NEW.Y := Types.COORDINATE(y_top_start_A);
    BOX_LIST(2).XY_NEW.Y := Types.COORDINATE(y_bottom_start_A);
    BOX_LIST(3).XY_NEW.Y := Types.COORDINATE(y_top_start_A);
    BOX_LIST(4).XY_NEW.Y := Types.COORDINATE(y_top_start_A);
  else                                     -- draw status_box at 'manual'
    BOX_LIST(1).XY_NEW.Y := Types.COORDINATE(y_top_start_M );
    BOX_LIST(2).XY_NEW.Y := Types.COORDINATE(y_bottom_start_M);
    BOX_LIST(3).XY_NEW.Y := Types.COORDINATE(y_top_start_M);
    BOX_LIST(4).XY_NEW.Y := Types.COORDINATE(y_top_start_M);
  end if;
--
--  Rendezvous with Graphics to draw new status_box
--
  --$TP(0079) Status.Update_Box rendezvous with Graphics start
  Graphics.Display.Move(MOVE_PRIORITY, BOX_LIST(Types.WORD_INDEX range box_start..box_end));
  --$TP(0080) Status.Update_Box rendezvous with Graphics end
--
--  Update status_box lists
--
  for I in Types.WORD_INDEX range box_start .. box_end loop
    BOX_LIST(I).XY_OLD := BOX_LIST(I).XY_NEW;
  end loop;
  --$TP(0081) Status.Update_Box end
end Update_Box;


procedure Display_Digits(NEXT_DATA : in out Types.WORD;
                         STAT      : STATUS_TYPE) is

-- A procedure which takes the DATA_OLD numbers, divides by 10 to get a single
-- digit.  That digit is used as an index into Shapes.NUMERIC, which holds
-- values to draw that number for Graphics.  It updates DATA_OLD in the process.

DIGIT      : Types.WORD;
STAT_X_LOC : Types.COORDINATE;

begin
  --$TP(0082) Status.Display_Digits start
--
-- Erase previous data
--
  for I in 1 .. Config.statistics_length loop
    DATA_OLD(STAT,I).COLOR := Graphics.background_color;
    WORK_LIST(Types.WORD_INDEX(I)) := DATA_OLD(STAT,I);
  end loop;
  --$TP(0083) Status.Display_Digits rendezvous with Graphics(1) start
  Graphics.Display.Move(MOVE_PRIORITY,WORK_LIST);
  --$TP(0084) Status.Display_Digits rendezvous with Graphics(1) end
```

```
--
-- Move new into old, then display
--
  STAT_X_LOC := base_x;
  for I in reverse 1 .. Config.statistics_length loop
    DIGIT := NEXT_DATA mod 10;                   -- get rightmost digit
    DATA_OLD(STAT,I).OBJECT:=(Shapes.PIXEL_MODE,Shapes.NUMERIC(INTEGER(DIGIT)));
    DATA_OLD(STAT,I).COLOR := Graphics.status_color;
    DATA_OLD(STAT,I).XY_NEW.X := STAT_X_LOC;
    STAT_X_LOC := STAT_X_LOC - Shapes.number_width; -- moving left
    WORK_LIST(Types.WORD_INDEX(I)) := DATA_OLD(STAT,I);
    NEXT_DATA := NEXT_DATA / 10;                 -- get next digit
  end loop;
  --$TP(0085) Status.Display_Digits rendezvous with Graphics(2) start
  Graphics.Display.MOVE(MOVE_PRIORITY,WORK_LIST);
  --$TP(0086) Status.Display_Digits rendezvous with Graphics(2) end
  --$TP(0087) Status.Display_Digits end
exception
  when others =>
    Debug_IO.Put_Line("Exception raised in Status.Display_Digits");
end Display_Digits;


--
-- body of UPDATE task
--
Begin
  Initialize;
  loop
    --$TP(0114) Status task start
    --$TP(0115) Status accept Signal start
    accept Signal;
    --$TP(0088) Status accept Signal end
    Interrupt_Control.Enable;
    begin                                        -- exception block
      loop
        Interrupt_Control.Disable;
        DISPLAY_REQUIRED := not MODE_DISPLAYED;
        NEXT_MODE := MODE;
        MODE_DISPLAYED := TRUE;
        Interrupt_Control.Enable;
        if DISPLAY_REQUIRED then                 -- update new status_box
          Update_Box(NEXT_MODE);
        end if;
        for I in STATUS_TYPE'first .. STATUS_TYPE'last loop
          Interrupt_Control.Disable;
          DISPLAY_REQUIRED := not STATUS_CONTROL(I).DISPLAYED;
          NEXT_DATA := STATUS_CONTROL(I).DATA;
          STATUS_CONTROL(I).DISPLAYED := TRUE;
          Interrupt_Control.Enable;
          if DISPLAY_REQUIRED then
          Display_Digits(NEXT_DATA,I);
        end if;
      end loop;
      Interrupt_Control.Disable;
      REQ_COUNT := REQ_COUNT - 1;
      exit when REQ_COUNT = 0;
      Interrupt_Control.Enable;
    end loop;
  --$TP(0089) Status task end
  exception
    when others => Debug_IO.Put_Line("Exception raised in Status task");
  end;
 end loop;
end Update_Type;

end Status;
```

```
--------------------------------------------------------------------------
--| UNIT:      Sync Package Spec.                                        --
--| Effects:  No current use.  Will provide greater synchronize in futr.--
--| Modifies: No global data is modified.                                --
--| Requires: No initialization is required.                             --
--| Raises:   No explicitly raised exceptions are propagated.            --
--| Engineer: T. Griest.                                                 --
--------------------------------------------------------------------------


--
--  package to synchronize clocks, will contain a task to call simulator
--  clock entries
--
with Types;

package Sync is
  type TIME_TYPE is new Types.WORD;
end Sync;
```

```
--------------------------------------------------------------------------
--| UNIT:     Target Package Spec.                                      --
--| Effects:  Provides structure for BDS Target management.             --
--| Modifies: No global data is modified.                               --
--| Requires: No initialization is required.                            --
--| Raises:   No explicitly raised exceptions are propagated.           --
--| Engineer: T. Griest.                                                --
--------------------------------------------------------------------------


--
--  package Target provides target tracking and display management
--
with Types;
with Config;

package Target is

  track_stack_size      : constant := 3928;
  track_data_stack_size : constant := 1506;

  subtype TARGET_ID_TYPE is Types.WORD_INDEX range 0..Config.max_targets;

  type TARGET_ITEM_TYPE is record     --| provides individual target information
    TARGET_ID        : TARGET_ID_TYPE;
    POSITION         : Types.POSITION_TYPE;
    TARGET_CLASS     : Types.TARGET_CLASS_TYPE;
  end record;

  type TARGET_LIST_TYPE is            --| list of all available targets items
            array(Types.WORD_INDEX range <>) of TARGET_ITEM_TYPE;

  type TARGET_MSG_TYPE is record      --| incoming message from Sensor
    NUM_TARGETS        : Types.WORD_INDEX;
    TARGET_LIST        : TARGET_LIST_TYPE(Types.TARGET_INDEX_TYPE);
  end record;

  type TARGET_STATUS_TYPE is record
    ACTIVE             : BOOLEAN;
    ENGAGED            : BOOLEAN;
    CLASS              : Types.TARGET_CLASS_TYPE;
  end record;
    for TARGET_STATUS_TYPE use record
      ACTIVE  at 0 range 0..0;
      ENGAGED at 0 range 1..1;
      CLASS   at 0 range 2..3;
    end record;

  type TARGET_DATA_TYPE is record
    STATUS             : TARGET_STATUS_TYPE;
    POSITION_NEW       : Types.POSITION_TYPE;
    POSITION_OLD       : Types.POSITION_TYPE;
  end record;

  type TARGET_DATA_LIST_TYPE is
    array(Types.TARGET_INDEX_TYPE)of TARGET_DATA_TYPE;

--
--  The TRACK task is used to control all of the target display information.
--  It accepts data from the Sensor and maintains it for the Rocket.Control
--  task.
--
  task type Track_Type is
    pragma PRIORITY(Config.track_priority);
  end Track_Type;
  for Track_Type'STORAGE_SIZE use INTEGER(Config.bytes_per_storage_unit *
                                                 track_stack_size);
  Track                : Track_Type;

--
-- The Track_Data task is used to buffer the most recent target list
-- from the Target.Track task and provide it to the Rocket.Control
-- task.  It also buffers new engagements from the Rocket.Control to
-- notify the Target.Track task that a new target has been engaged.
-- Note that only one new target can be engaged every update interval.
-- If the NEXT_ENGAGE parameter is 0, this is an invalid TARGET_ID, and
-- implies that no new target is engaged.
--
  task type Track_Data_Type is
    entry Put(DATA           : in  TARGET_DATA_LIST_TYPE; --| put new list
              NEXT_ENGAGE    : out TARGET_ID_TYPE;        --| get new engagement
              NEXT_DISENGAGE : out TARGET_ID_TYPE); --| and disengagement

    entry Get(DATA           : out TARGET_DATA_LIST_TYPE; --| get new list
              NEXT_ENGAGE    : in  TARGET_ID_TYPE;        --| put new engagement
              NEXT_DISENGAGE : in  TARGET_ID_TYPE);       --| and disengagement
    pragma PRIORITY(Config.track_data_priority);
  end Track_Data_Type;
  for Track_Data_Type'STORAGE_SIZE use INTEGER(Config.bytes_per_storage_unit *
                                                 track_data_stack_size);
  Track_Data           : Track_Data_Type;


end Target;   -- package specification
```

```
-----------------------------------------------------------------------
--| UNIT:     Target Package Body.                                   --
--| Effects:  Provides structure for BDS Target management.          --
--| Modifies: No global data is modified.                            --
--| Requires: No initialization is required.                         --
--| Raises:   No explicitly raised exceptions are propagated.        --
--| Engineer: T. Griest.                                             --
-----------------------------------------------------------------------


with Simulate;    pragma ELABORATE(Simulate);
--
--  package Target provides target tracking and display management
--
package body Target is

--
--  The TRACK task is used to control all of the target display information.
--  It gets data from the Sensor and maintains it for the Rocket.Control
--  task.
--
  task body Track_Type is separate;

--
-- The Track_Data task is used to buffer the most recent target list
-- from the Target.Track task and provide it to the Rocket.Control
-- task.  It also buffers new engagements from the Rocket.Control to
-- notify the Target.Track task that a new target has been engaged.
-- Note that only one new target can be engaged every update interval.
-- If the NEXT_ENGAGE parameter is 0, this is an invalid TARGET_ID, and
-- implies that no new target is engaged.
--
  task body Track_Data_Type is separate;

end Target;   -- package body
```

```
---------------------------------------------------------------------------
--| UNIT:     Targ_Sup Task Body Subunit.                                --
--| Effects:  Provides Simulator motion control for all targets.         --
--| Modifies: No global data is modified.                                --
--| Requires: No initialization is required.                             --
--| Raises:   TARGET_CREATE_ERROR if told to create when max exceeded.   --
--| Engineer: M. Sperry.                                                 --
---------------------------------------------------------------------------


with Calendar;
with Debug_IO;
with Unchecked_Conversion;
with Low_Level_IO;
with Time_Stamp;
use Low_Level_IO;

pragma ELABORATE(Calendar, Debug_IO, Time_Stamp);

separate (Simulate.Sensor)

task body Targ_Sup_Type is

-- A task which sends a list to the caller describing new targets and
-- targets which have been destroyed. They are described by not being on
-- the list. Note that new targets are created first and then those
-- that need to be destroyed are processed. This task is timed so that
-- the list is ready only during 100 millisecond intervals.

use Calendar;                                     --for visibility to "-"
use Types;                                        --for visibility to "/" etc.

type MOTION_REC is record
  OFFSET : Types.METERS;
  COUNT  : Types.WORD;
end record;

start_y             : constant Types.METERS := 3960.0;
start_z             : constant Types.METERS := 0.0;
distance_per_report : constant Types.METERS := 2.0;  -- meters in Y per report
safety_factor       : constant Types.METERS := 48.0; -- inner border limits
min_dir_time        : constant Types.WORD := 20; -- direction travelling time

TARGET_LIMIT        : Types.WORD_INDEX := 5;
TARGET_COUNT        : Types.WORD_INDEX := 0;         -- local count of targets
TARGET_COUNTER      : Types.WORD_INDEX;              -- Target index for array
CLASS               : Types.TARGET_CLASS_TYPE;       -- type of class for target
TARGET_ID           : Types.TARGET_INDEX_TYPE := 1;  -- target id used
MOTION              : array(Types.TARGET_INDEX_TYPE'first..
                            Types.TARGET_INDEX_TYPE'last) of MOTION_REC;
TEMP                : Types.POSITION_TYPE;           -- for fixed compiler bug
```

```
TARGET_CREATE_ERROR : EXCEPTION;
START_TIME          : Calendar.TIME;
DELAY_PERIOD        : DURATION;


function WORD_TO_METERS is new Unchecked_Conversion(Types.WORD,Types.METERS);
function BYTE_TO_WORD is new Unchecked_Conversion(Low_Level_IO.BYTE,Types.WORD);


function RND return Types.WORD is

counter_two         : constant Low_Level_IO.PORT_ADDRESS := 16#42#;

DATA                : Low_Level_IO.BYTE;
RANDOM_NUMBER       : Types.WORD;

begin
  Low_Level_IO.Receive_Control(counter_two,DATA);
  RANDOM_NUMBER := BYTE_TO_WORD(DATA);
  return RANDOM_NUMBER;
end RND;
pragma INLINE(RND);



procedure Initialize is

-- A procedure used to intialize the Simulator's data base, and start the
-- channel 2 counter which is used to determine the time it takes for a target
-- to switch directions as well as which direction to turn.

timer_cntrl : constant Low_Level_IO.PORT_ADDRESS := 16#43#;
counter_two : constant Low_Level_IO.PORT_ADDRESS := 16#42#;
intialize   : constant Low_Level_IO.BYTE := 16#B2#;          -- mode 2, channel 2
start_count : constant Low_Level_IO.BYTE := 16#FF#;

begin
  Low_Level_IO.Send_Control(timer_cntrl,intialize);
  Low_Level_IO.Send_Control(counter_two,start_count);
  Low_Level_IO.Send_Control(counter_two,start_count);
  for I in Types.TARGET_INDEX_TYPE'first..Types.TARGET_INDEX_TYPE'last loop
    Simulate.TARGETS(I).ACTIVE := FALSE;
    MOTION(I).OFFSET := WORD_TO_METERS(RND / 16 - 8);
    MOTION(I).COUNT := RND + min_dir_time;
  end loop;
  CLASS := Types.TARGET_CLASS_TYPE'FIRST;
end Initialize;


procedure Create_New_Target is
```

```
-- A procedure used to create a new target.  The first usable ID is taken
-- to be the new targets ID.  Also the target class is changed for every create.

FOUND  : BOOLEAN := FALSE;
ID     : Types.TARGET_INDEX_TYPE;
OLD_ID : Types.TARGET_INDEX_TYPE;

begin
  --$TP(0090) Targ_Sup.Create start
  ID := TARGET_ID;
  OLD_ID := TARGET_ID;
  while not FOUND loop
    if not Simulate.TARGETS(ID).ACTIVE then
      Simulate.TARGETS(ID).ACTIVE := TRUE;
      if CLASS = Types.TARGET_CLASS_TYPE'LAST then
        CLASS := Types.TARGET_CLASS_TYPE'FIRST;
      else
        CLASS := Types.TARGET_CLASS_TYPE'SUCC(CLASS);
      end if;
      Simulate.TARGETS(ID).TARGET_CLASS := CLASS;
      Simulate.TARGETS(ID).POSITION.X := WORD_TO_METERS(RND * 15 + 10) * 8;
      Simulate.TARGETS(ID).POSITION.Y := start_y;
      Simulate.TARGETS(ID).POSITION.Z := start_z;
      TARGET_COUNT := TARGET_COUNT + 1;
      TARGET_ID := ID;                 -- keep rollover count of TARGET_ID's
      FOUND := TRUE;
      exit;
    else
      ID := ID + 1;
      if ID > Config.max_targets then
        ID := 1;
        if OLD_ID = ID then
          raise TARGET_CREATE_ERROR;
        end if;                        -- no more room check
      end if;                          -- wrap ID check
    end if;                            -- ACTIVE check
  end loop;
  --$TP(0091) Targ_Sup.Create end
exception
  when others =>
    Debug_IO.Put_Line("Exception raised in Targ_Sup.Create_New_Target.");
end Create_New_Target;


--
-- Targ_Sup task body
--
begin
  Initialize;
  -- First take the time.
  START_TIME := Calendar.Clock;
```

```
loop
  --$TP(0092) Targ_Sup task start
  START_TIME := START_TIME + Config.interval;
  -- Then check number of Targets; if less than maximum, then add a new
  -- Target to the list.
  if TARGET_COUNT < TARGET_LIMIT then
    Create_New_Target;
  end if;
  -- Then move each target.
  for ID in Types.TARGET_INDEX_TYPE'first..Types.TARGET_INDEX_TYPE'last loop
    if Simulate.TARGETS(ID).ACTIVE then
      Simulate.TARGETS(ID).POSITION.Y := Simulate.TARGETS(ID).POSITION.Y -
                                                  distance_per_report;
      Simulate.TARGETS(ID).POSITION.X := Simulate.TARGETS(ID).POSITION.X +
                                                  MOTION(ID).OFFSET;
      -- Check for hitting against boundaries.
      if Simulate.TARGETS(ID).POSITION.X > Config.meters_in_battle_area then
        Simulate.TARGETS(ID).POSITION.X := Config.meters_in_battle_area -
                                                  safety_factor;
        MOTION(ID).OFFSET := -(MOTION(ID).OFFSET);--move in opposite direction
        MOTION(ID).COUNT := RND + min_dir_time;
      elsif Simulate.TARGETS(ID).POSITION.X < safety_factor then
        Simulate.TARGETS(ID).POSITION.X := safety_factor;
        MOTION(ID).OFFSET := -(MOTION(ID).OFFSET);--move in opposite direction
        MOTION(ID).COUNT := RND + min_dir_time;
      end if;
      MOTION(ID).COUNT := MOTION(ID).COUNT - 1;
      if MOTION(ID).COUNT = 0 then  -- time expired, possibly change direction
        MOTION(ID).COUNT := RND + min_dir_time;
        MOTION(ID).OFFSET := WORD_TO_METERS(RND / 16 - 8);
      end if;                       -- timeout on direction check
    end if;                         -- ACTIVE check
  end loop;
  -- Then see if any targets made it to the enemy line.
  -- These targets are no longer the concern of the BDS. They
  -- are deleted from the list.
  for I in Types.TARGET_INDEX_TYPE loop
    if Simulate.TARGETS(I).ACTIVE then
      if Simulate.TARGETS(I).POSITION.Y < 40.0 then
        TARGET_COUNT := TARGET_COUNT - 1;
        Simulate.TARGETS(I).ACTIVE := FALSE;
      end if;
    end if;
  end loop;
  -- Finally move the list into the target list kept by the target spec.
  --$TP(0093) Targ_Sup accept Next_Target_Msg start
  accept Next_Target_Msg(DATA : out Target.TARGET_MSG_TYPE) do
    TARGET_COUNTER := 0;
    for I in Types.TARGET_INDEX_TYPE loop
      if Simulate.TARGETS(I).ACTIVE then
        TARGET_COUNTER := TARGET_COUNTER + 1;
```

```
          TEMP := Simulate.TARGETS(1).POSITION;  -- fixed compiler code bug
          DATA.TARGET_LIST(TARGET_COUNTER).POSITION := TEMP;
          DATA.TARGET_LIST(TARGET_COUNTER).TARGET_CLASS :=
                                            Simulate.TARGETS(1).TARGET_CLASS;
          DATA.TARGET_LIST(TARGET_COUNTER).TARGET_ID := 1;
        end if;
      end loop;
      TARGET_COUNT := TARGET_COUNTER;
      DATA.NUM_TARGETS := TARGET_COUNTER;
    end Next_Target_Msg;
    --$TP(0094) Targ_Sup accept Next_Target_Msg end
    DELAY_PERIOD := START_TIME - Calendar.Clock;
    if DELAY_PERIOD < 0.0 then
      START_TIME := Calendar.Clock;
    end if;
    --$TP(0095) Targ_Sup end
    delay DELAY_PERIOD;
  end loop;
--  accept Clock(Time : in  Sync.TIME_TYPE);   --TBD
end Targ_Sup_Type;
```

```
-----------------------------------------------------------------------
--| UNIT:     Track Task Body Subunit.                                --
--| Effects:  Provides all target tracking and display for BDS.       --
--| Modifies: No global data is modified.                             --
--| Requires: No initialization is required.                          --
--| Raises:   No explicitly raised exceptions are propagated.         --
--| Engineer: T. Griest.                                              --
-----------------------------------------------------------------------


with Graphics;
with Shapes;
with Interrupt_Control;
with Grid_to_Pixel;
with Simulate;
with Debug_IO;
with Status;
with Time_Stamp;
pragma ELABORATE(Graphics,Shapes, Interrupt_Control,Grid_to_Pixel,
                 Simulate, Debug_IO, Status, Time_Stamp);

separate (Target)
--
--  The TRACK task is used to control all of the target display information.
--  It accepts data from the Sensor and maintains it for the Rocket.Control
--  task.
--
task body Track_Type is
  use Types;
  package Sensor renames Simulate.Sensor;  -- make simulation transparent
  use Types;                    -- for operators only
  TARGET_MSG       : TARGET_MSG_TYPE;
  MOVE_TARGETS     : Graphics.MOVE_LIST_TYPE(Types.TARGET_INDEX_TYPE);
  MOVE_INDEX       : Types.WORD_INDEX;
  DESTROYED        : Types.WORD;
  CREATED          : Types.WORD;
  PIXEL_POINT      : Shapes.PIXEL;
  TARGETS          : TARGET_DATA_LIST_TYPE;
  MSG_INDEX        : Types.WORD_INDEX;
  NEXT_ENGAGED     : Types.WORD_INDEX;    -- 0 if no new engagement
  NEXT_DISENGAGED  : Target.TARGET_ID_TYPE;-- keep track of disengagements
  COLOR            : Graphics.COLOR_TYPE;
  ENGAGE_FLAG      : BOOLEAN;
  CLASS            : Types.TARGET_CLASS_TYPE;
  POSITION         : Types.POSITION_TYPE;  -- temp for making changes
begin
--
--  INITIALIZATION
--
  for I in TARGETS'range loop
    TARGETS(I).STATUS := (FALSE,FALSE,UNKNOWN);  -- init to default
```

```
end loop;

loop
  --$TP(0096) Track task start
  --$TP(0097) Track rendezvous with Targ_Sup start
  Sensor.Targ_Sup.Next_Target_Msg(TARGET_MSG);
  --$TP(0098) Track rendezvous with Targ_Sup end
--
-- Zero out counters
--
  CREATED := 0;
  DESTROYED := 0;
--
-- Maintain history information.
--
-- Go through each target to examine its new status
--
  MSG_INDEX := 1;
  MOVE_INDEX := 0;
  for TARGET_ID in TARGETS'RANGE loop
    if TARGETS(TARGET_ID).STATUS.ACTIVE then
      if MSG_INDEX > TARGET_MSG.NUM_TARGETS or else
         TARGET_MSG.TARGET_LIST(MSG_INDEX).TARGET_ID
                                    /= TARGET_ID then -- target destroyed
--
-- Target has been destroyed, keep local accumulation of destroyed
-- targets, and add to list for Display task to erase target.
--
        DESTROYED := DESTROYED + 1;
-- mark as inactive     (ACTIVE => FALSE, ENGAGED => FALSE, CLASS => UNKNOWN)
        TARGETS(TARGET_ID).STATUS := (FALSE, FALSE, Types.UNKNOWN);
        MOVE_INDEX := MOVE_INDEX + 1;
        PIXEL_POINT := Grid_To_Pixel(TARGETS(TARGET_ID).POSITION_NEW);
        COLOR := Graphics.background_color;
        MOVE_TARGETS(MOVE_INDEX) := (PIXEL_POINT,
                                     PIXEL_POINT,
                                     (Shapes.PIXEL_MODE,Shapes.TARGET),
                                     COLOR);
      else                                          -- move the target
--
-- Found a current existing target in the latest sensor report,
-- update target information and add it to move list.
--
        POSITION := TARGET_MSG.TARGET_LIST(MSG_INDEX).POSITION;
        MOVE_INDEX := MOVE_INDEX + 1;
        CLASS := TARGETS(TARGET_ID).STATUS.CLASS;
        ENGAGE_FLAG := TARGETS(TARGET_ID).STATUS.ENGAGED;
        COLOR := Graphics.target_color(CLASS, ENGAGE_FLAG);
        MOVE_TARGETS(MOVE_INDEX) :=
                 (XY_OLD => Grid_to_Pixel(TARGETS(TARGET_ID).POSITION_NEW),
                  XY_NEW => Grid_to_Pixel(POSITION),
                OBJECT => (Shapes.PIXEL_MODE,Shapes.TARGET),
                COLOR  => COLOR
               );
        TARGETS(TARGET_ID).POSITION_OLD := TARGETS(TARGET_ID).POSITION_NEW;
        TARGETS(TARGET_ID).POSITION_NEW := POSITION;
        MSG_INDEX := MSG_INDEX + 1;
      end if;        -- new/old target check
    else                            -- this target wasn't previously active
      if MSG_INDEX <= TARGET_MSG.NUM_TARGETS and then
         TARGET_MSG.TARGET_LIST(MSG_INDEX).TARGET_ID
                                           = TARGET_ID then -- new target
--
--  New Target has been created, set status and put it on display
--
        CREATED := CREATED + 1;
 -- mark as active
        TARGETS(TARGET_ID).STATUS :=
               (TRUE,          -- ACTIVE
                FALSE,         -- Engaged
                TARGET_MSG.TARGET_LIST(MSG_INDEX).TARGET_CLASS);  -- class
        TARGETS(TARGET_ID).POSITION_OLD :=      -- set both old and new
                          TARGET_MSG.TARGET_LIST(MSG_INDEX).POSITION;
        TARGETS(TARGET_ID).POSITION_NEW :=
                          TARGET_MSG.TARGET_LIST(MSG_INDEX).POSITION;
        MOVE_INDEX := MOVE_INDEX + 1;
        CLASS := TARGETS(TARGET_ID).STATUS.CLASS;
        ENGAGE_FLAG := TARGETS(TARGET_ID).STATUS.ENGAGED;
        COLOR := Graphics.target_color(CLASS, ENGAGE_FLAG);
        MOVE_TARGETS(MOVE_INDEX) :=
               (XY_OLD => Grid_to_Pixel(TARGETS(TARGET_ID).POSITION_OLD),
                XY_NEW => Grid_to_Pixel(TARGETS(TARGET_ID).POSITION_NEW),
                OBJECT => (Shapes.PIXEL_MODE,Shapes.TARGET),
                COLOR  => COLOR
               );
        MSG_INDEX := MSG_INDEX + 1;
      end if;                          -- end of new target check
    end if;                            -- active check
  end loop;
--
--  Now update status if any created or destroyed
--
  if CREATED /= DESTROYED or DESTROYED > 0 then
    Interrupt_Control.Disable;
    Status.STATUS_CONTROL(Status.TRACKED).DATA :=
       Status.STATUS_CONTROL(Status.TRACKED).DATA + (CREATED - DESTROYED);
    Status.STATUS_CONTROL(Status.TRACKED).DISPLAYED := FALSE;
    Status.STATUS_CONTROL(Status.DESTROYED).DATA :=
       Status.STATUS_CONTROL(Status.DESTROYED).DATA + DESTROYED;
    Status.STATUS_CONTROL(Status.DESTROYED).DISPLAYED := FALSE;
    Status.REQ_COUNT := Status.REQ_COUNT + 1;
    if Status.REQ_COUNT = 1 then
```

```
      --$TP(0099) Track rendezvous with Status start
        Status.Update.Signal;
      --$TP(0100) Track rendezvous with Status end
      end if;
      Interrupt_Control.Enable;
    end if;
    --$TP(0101) Track rendezvous with Track_Data start
    Target.Track_Data.Put(TARGETS,NEXT_ENGAGED,NEXT_DISENGAGED);
                                                 -- send copy to Rocket.Control
    --$TP(0102) Track rendezvous with Track_Data end
    if NEXT_ENGAGED > 0 then
      TARGETS(NEXT_ENGAGED).STATUS.ENGAGED := TRUE;   -- set engaged
    end if;
    if NEXT_DISENGAGED > 0 then
      TARGETS(NEXT_DISENGAGED).STATUS.ENGAGED := FALSE;
    end if;
    --$TP(0103) Track rendezvous with Graphics start
    Graphics.Display.Move(Graphics.LOW, MOVE_TARGETS(1..MOVE_INDEX));
    --$TP(0104) Track rendezvous with Graphics end
    --$TP(0105) Track task end
  end loop;
exception
  when others =>
    Debug_IO.Put_Line("TRACK termination due to exception.");
end Track_Type;
```

```
-----------------------------------------------------------------------
--| UNIT:     Track_Data Task Subunit.                                --
--| Effects:  Provides buffering of target tracking data between the  --
--|           Track task and the Control task for rocket engagement.  --
--| Modifies: No global data is modified.                             --
--| Requires: No initialization is required.                          --
--| Raises:   No explicitly raised exceptions are propagated.         --
--| Engineer: T. Griest.                                              --
-----------------------------------------------------------------------

with Time_Stamp;
with Interrupt_Control;
pragma ELABORATE(Time_Stamp,Interrupt_Control);

separate (Target)
--
-- The Track_Data task is used to buffer the most recent target list
-- from the Target.Track task and provide it to the Rocket.Control
-- task.  It also buffers new engagements from the Rocket.Control to
-- notify the Target.Track task that a new target has been engaged.
-- Note that only one new target can be engaged every update interval.
-- If the NEXT_ENGAGE parameter is 0, this is an invalid TARGET_ID, and
-- implies that no new target is engaged.
--

task body Track_Data_Type is
  use Types;

  BUFFERED_DATA      : Target.TARGET_DATA_LIST_TYPE;
  BUFFERED_ENGAGE    : Target.TARGET_ID_TYPE;
  BUFFERED_DISENGAGE : Target.TARGET_ID_TYPE;
  DATA_COUNT         : Types.WORD := 0;
begin
--
--  Initialize local copy of data
--  initialize all target status to:
--            (ACTIVE => FALSE, ENGAGED => FALSE, CLASS => UNKNOWN)
--
  BUFFERED_ENGAGE := 0;                      -- default is no new engagement
  for I in BUFFERED_DATA'range loop
    BUFFERED_DATA(I).STATUS := (FALSE, FALSE, Types.UNKNOWN);
  end loop;
  loop
    select
      accept Put(DATA           : in  TARGET_DATA_LIST_TYPE;
                 NEXT_ENGAGE    : out TARGET_ID_TYPE;
                 NEXT_DISENGAGE : out TARGET_ID_TYPE) do
        --$TP(0106) Trackdat accept Put start
        Interrupt_Control.Disable;
        BUFFERED_DATA := DATA;
        Interrupt_Control.Enable;
        NEXT_ENGAGE := BUFFERED_ENGAGE;
        NEXT_DISENGAGE := BUFFERED_DISENGAGE;
        DATA_COUNT := 1;
        --$TP(0107) Trackdat accept Put end
      end Put;
    or
      when DATA_COUNT > 0 =>
      accept Get(DATA           : out TARGET_DATA_LIST_TYPE;
                 NEXT_ENGAGE    : in  TARGET_ID_TYPE;
                 NEXT_DISENGAGE : in  TARGET_ID_TYPE) do
        --$TP(0108) Trackdat accept Get start
        Interrupt_Control.Disable;
        DATA := BUFFERED_DATA;
        Interrupt_Control.Enable;
        BUFFERED_ENGAGE := NEXT_ENGAGE;
        BUFFERED_DISENGAGE := NEXT_DISENGAGE;
        DATA_COUNT := 0;
        --$TP(0109) Trackdat accept Get end
      end Get;
    end select;
  end loop;
end Track_Data_Type;
```

```
---------------------------------------------------------------------
--| UNIT:     Traject Function Spec.                                --
--| Effects:  Computes rocket motion based on previous motion and  --
--|           aimpoints received in guidance messages.             --
--| Modifies: No global data is modified.                          --
--| Requires: No initialization is required.                       --
--| Raises:   No explicitly raised exceptions are propagated.      --
--| Engineer: T. Griest.                                           --
---------------------------------------------------------------------


--
--  Traject: Is the trajectory planner for taking an Azimuth, Elevation
--           X,Y,Z position and constant velocity and producing a new
--           position
with Types;

function Traject(POSC : Types.POSITION_TYPE; AIMPOINT : Types.AIMPOINT_TYPE)
      return Types.POSITION_TYPE;
```

```
-------------------------------------------------------------------
--| UNIT:     Traject Function Body.                             --
--| Effects:  Computes rocket motion based on previous motion and --
--|           aimpoints received in guidance messages.           --
--| Modifies: No global data is modified.                        --
--| Requires: No initialization is required.                     --
--| Raises:   No explicitly raised exceptions are propagated.    --
--| Engineer: T. Griest.                                         --
-------------------------------------------------------------------



--
--  Traject: Is the trajectory planner for taking an Azimuth, Elevation
--           X,Y,Z position and constant velocity and producing a new
--           position
with Math;
with Time_Stamp;
pragma ELABORATE(Math, Time_Stamp);

function Traject(POS0 : Types.POSITION_TYPE; AIMPOINT : Types.AIMPOINT_TYPE)
return Types.POSITION_TYPE is
  use Types;                              -- for operators
  velocity    : constant := 20;           -- meters per interval

  SCALE       : Types.LONG_FIXED;
  X0, Y0, Z0  : Types.LONG_FIXED;
  X0_SQ       : Types.LONG_FIXED;         -- X0 ** 2
  ELEVATION   : Types.BAM;
  AZIMUTH     : Types.BAM;
  DIRECTION_X : INTEGER;
  DIRECTION_Y : INTEGER;
  DIRECTION_Z : INTEGER;

  NEW_POS    : Types.POSITION_TYPE;
begin
  --$TP(0110) Traject start
  if AIMPOINT.AZIMUTH >= 0 then
    DIRECTION_Y := 1;
 --   AZIMUTH := AIMPOINT.AZIMUTH;
  else
    DIRECTION_Y := -1;
--    AZIMUTH := -AIMPOINT.AZIMUTH;
  end if;

  if abs AIMPOINT.AZIMUTH <= 16384 then
    DIRECTION_X := 1;
  else
    DIRECTION_X := -1;
--    AZIMUTH := (32767 - AZIMUTH) + 1;
  end if;
```

```
--  if AIMPOINT.ELEVATION >= 0 then
--    DIRECTION_Z := 1;
--    ELEVATION := AIMPOINT.ELEVATION;
--  else
--    DIRECTION_Z := -1;
--    ELEVATION := -AIMPOINT.ELEVATION;
--  end if;


  YO := 1.0;
  XO := Math.Tan(AIMPOINT.AZIMUTH);
  if XO = 0.0 then
    XO := Types.sqrt_large_number;
  else
    XO := Types.LONG_FIXED(Types.LONG_FIXED(1.0)/XO);
  end if;
  XO_SQ := Types.LONG_FIXED(XO * XO);
  ZO := Types.LONG_FIXED(Math.Tan(AIMPOINT.ELEVATION) *
                      Math.Sqrt(XO_SQ + Types.LONG_FIXED(1.0)));

  if ZO > Types.sqrt_large_number then
    ZO := Types.sqrt_large_number;
  end if;
  SCALE := Math.Sqrt(YO + XO_Sq + Types.LONG_FIXED(ZO*ZO));
  NEW_POS.X := Types.METERS(velocity * DIRECTION_X * abs XO / SCALE) + POSO.X;
  NEW_POS.Y := Types.METERS(Types.LONG_FIXED(velocity) /
                                          SCALE)*DIRECTION_Y + POSO.Y;
  NEW_POS.Z := Types.METERS(velocity * ZO / SCALE) + POSO.Z;
  --$TP(0111) Traject end
  return NEW_POS;
end Traject;
```

```
-----------------------------------------------------------------------
--| UNIT:     Types Package Spec.                                     --
--| Effects:  Provides general purpose data types.                    --
--| Modifies: No global data is modified.                             --
--| Requires: No initialization is required.                          --
--| Raises:   No explicitly raised exceptions are propagated.         --
--| Engineer: T. Griest.                                              --
-----------------------------------------------------------------------


-- Date     : 10-10-88
-- Purpose :
--   The purpose of Types is to provide common project specific data types that
-- are not related to any particular application area.

with Config;

package Types is

type WORD is range -32768 .. 32767;
  for WORD'size use 16;

type WORD_INDEX is range 0 .. 32767;
  for WORD_INDEX'size use 16;

subtype WORD_COUNT is WORD range 0 .. 32767;

subtype ROCKET_INDEX_TYPE is WORD_INDEX range 1..Config.max_rockets;
subtype TARGET_INDEX_TYPE is WORD_INDEX range 1..Config.max_targets;

subtype COORDINATE is Types.WORD;
subtype REL_COORDINATE is Types.WORD;


type METERS is delta 0.125 range -Config.meters_in_battle_area ..
                                           Config.meters_in_battle_area;

type LONG_FIXED is delta 0.015625 range -33_554_432.0..33_554_431.0;
for LONG_FIXED'size use 32;

sqrt_large_number : constant := 2508.0;  -- approx sqrt(LONG_FIXED'last)/4

type POSITION_TYPE is record             --| for absolute position
  X          : METERS;                   --| assume battlefield oriented ENU
  Y          : METERS;
  Z          : METERS;
end record;

type POSITION_PAIR_TYPE is record   --| X,Y,Z in meters
  TARGET_OLD    : Types.POSITION_TYPE;
  TARGET_NEW    : Types.POSITION_TYPE;
  ROCKET_OLD    : Types.POSITION_TYPE;
  ROCKET_NEW    : Types.POSITION_TYPE;
end record;

type BAM is range -32768 .. 32767;      --| binary angle measurement 32768/180
                                        --| East North Up origins (0)

type AIMPOINT_TYPE is record
  AZIMUTH    : BAM;
  ELEVATION : BAM;
end record;

subtype DISTANCE is METERS range 0.0 .. Config.meters_in_battle_area;

--
--  T80 - Main Battle Tank
--  SA9  - GASKIN surface to air missle launcher
--  BMP2 - Infantry Combat Vehicle
type TARGET_CLASS_TYPE is (UNKNOWN, T80, SA9, BMP2);

end Types;
```

20 Appendix I - Distributed Runtime Source Code

The source code for the distributed runtime uses an 8086 family assembly language code. It is divided into modules which implement the major functional areas. These include: program linkage, runtime routines, network setup, network I/O, and vendor runtime interface.

```
        .XLIST
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;   FILE:   DA_DEF.ASM                                                 ;
;   Distributed Ada - Definitions                                      ;
;                                                                      ;
;   Definitions for system values                                      ;
;                                                                      ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;; Distribution and Copyright ;;;;;;;;;;;;;;;;;;;;;;;
;   Derivation   : LabTek Distributed Ada V1.0                         ;
;                                                                      ;
;   This Distributed Ada Runtime inherits the LabTek copyright.        ;
;   The following copyright must be included in all software           ;
;   utilizing this Ada Runtime.                                        ;
;                                                                      ;
;   Copyright 1989 by LabTek Corporation, Woodbridge, CT, USA          ;
;                                                                      ;
;   Permission to use, copy, modify, and distribute this               ;
;   software and its documentation for any purpose and without         ;
;   fee is hereby granted, provided that the above copyright           ;
;   notice appear in all copies and that both that copyright           ;
;   notice and this permission notice appear in supporting             ;
;   documentation, and that the name of LabTek not be used in          ;
;   advertising or publicity pertaining to distribution of the         ;
;   software without specific, written prior permission.              ;
;   LabTek makes no representations about the suitability of           ;
;   this software for any purpose.  It is provided "as is"             ;
;   without express or implied warranty.                               ;
;                                                                      ;
;;;;;;;;;;;;;;;;;; Disclaimer ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                                                      ;
;   This software and its documentation are provided "AS IS" and       ;
;   without any expressed or implied warranties whatsoever.            ;
;   No warranties as to performance, merchantability, or fitness       ;
;   for a particular purpose exist.                                    ;
;                                                                      ;
;   In no event shall any person or organization of people be          ;
;   held responsible for any direct, indirect, consequential           ;
;   or inconsequential damages or lost profits.                        ;
;                                                                      ;
;;;;;;;;;;;;;;;;;; END-PROLOGUE ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


;
; NETWORK MESSAGE CONTROL FIELD VALUES
;
destination_addr        equ     0       ; Ethernet address of receiver
source_addr             equ     6       ; Ethernet address of sender
RCP                     equ     12      ; Receive Control Pointer Offset
priority                equ     14      ; Ada priority (not used at present)
sequence                equ     16      ; Packet Sequence Counter
data_length             equ     18      ; length of data field
```

```
data_field              equ     20      ; Data field(s)

;
;  Offsets from Receive Pointer (AT BUFFER+RCP) to various fields
;
DEF_cmd_offset          equ     0       ; command field
DEF_pid_offset          equ     2       ; processor ID
DEF_tid_offset          equ     4       ; task ID
DEF_eid_offset          equ     6       ; entry ID
DEF_reply_offset        equ     8       ; Reply message
;
;  Offset from list pointer to next node pointers
;
DEF_Next_Ptr            equ     2       ; offset to next pointer in buffer
;
;   TCB Offsets
;
DEF_TCB_EID             equ     6       ; Offset to Entries in TCB
DEF_TCB_ENTRY_SIZE      equ     4       ; four bytes per Entry
DEF_TCB_REPLY           equ     14      ; offset to rendezvous reply pointer

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; PROCESSOR / TASK / ENTRY IDs ;
;  Note: PIDs increment by 6,  ;
;        TIDs and EIDs by 2.   ;
; TASK IDs are unique.         ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
DEF_alpha_address       struc
;                        db      2, 96,140, 68, 82,  9
                        db      2, 96,140, 71, 99, 85
DEF_alpha_address       ends

DEF_bravo_address       struc
;                       db      2, 96,140, 58H, 35H, 68H
                        db      2, 96,140, 48H, 51H, 60H
DEF_bravo_address       ends


DEF_pid_alpha               equ     0       ; Primary Processor
DEF_pid_bravo               equ     6       ; Secondary Processor


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;   COMMANDS received via messages
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
DEF_elab_begin          equ     0
DEF_elab_end            equ     2
DEF_request_entry       equ     4
DEF_accept_complete     equ     6
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;   Parameter Passing convention to runtime network msg routines.
;
;  Standard Call Frame:
;
DEF_task_id         equ     6               ; BP + 6
DEF_TCP             equ     8               ; BP + 8 Transmit control pointer
DEF_Entry           equ     8               ; BP + 8 (also used for entry ID)
DEF_parameters      equ     10              ; BP + 10

DEF_param_count     equ     0       ; relative to parameters (In/Out)

DEF_param_offset    equ     0       ; followed by segment
DEF_param_segment   equ     2
DEF_param_length    equ     4       ; relative to offset
DEF_param_descriptor equ    6       ; # of bytes per parameter

DEF_local_task_id   equ     6       ; buffer offset for task_id

DEF_local_entry_id  equ     6       ; offset from BP in local end rendezvous


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;  REMOTE ENTRY CALL                                                      ;
;               *************************                                 ;
;  Low address  *        TASK ID        *    BP+6                         ;
;               *************************                                 ;
;               *       CONTROL PTR     *     +8                          ;
;               *************************                                 ;
;               *  IN Parameter Count   *     etc.                        ;
;               *************************                                 ;
;               *   IN PARM1 OFFSET     *                                 ;
;               *************************                                 ;
;               *   IN PARM1 SEGMENT    *                                 ;
;               *************************                                 ;
;               *   IN PARM1 LENGTH     *                                 ;
;               *************************                                 ;
;                        ...                                              ;
;                        ...                                              ;
;                        ...                                              ;
;               *************************                                 ;
;               *   IN PARMn OFFSET     *                                 ;
;               *************************                                 ;
;               *   IN PARMn SEGMENT    *                                 ;
;               *************************                                 ;
;               *   IN PARMn LENGTH     *                                 ;
;               *************************                                 ;
;                                                                         ;
;                                                                         ;
;    Note: Out parameters are accessible via the buffer decriptor         ;
;    pointed to by (BX).                                                  ;
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; REMOTE END RENDEZVOUS                                                ;
;               *************************                              ;
; Low address   *        TASK ID        *   BP+6                       ;
;               *************************                              ;
;               *       CONTROL PTR     *     +8                       ;
;               *************************                              ;
;               * OUT Parameter Count   *     etc.                     ;
;               *************************                              ;
;               * OUT PARM1 OFFSET      *                              ;
;               *************************                              ;
;               * OUT PARM1 SEGMENT     *                              ;
;               *************************                              ;
;               * OUT PARM1 LENGTH      *                              ;
;               *************************                              ;
;                       ...                                            ;
;                       ...                                            ;
;                       ...                                            ;
;               *************************                              ;
;               * OUT PARMn OFFSET      *                              ;
;               *************************                              ;
;               * OUT PARMn SEGMENT     *                              ;
;               *************************                              ;
;               * OUT PARMn LENGTH      *                              ;
;               *************************                              ;
;                                                                      ;
;                                                                      ;
;   Note: Out parameters are accessible via the buffer decriptor      ;
;   pointed to by (BX).                                                ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; SELECT                                                               ;
;               *************************                              ;
; Low address   *        TASK ID        *   BP+6                       ;
;               *************************                              ;
;               * NUMBER OF ENTRIES     *     +8                       ;
;               *************************                              ;
;               * ENTRY 1               *     etc.                     ;
;               *************************                              ;
;               * ENTRY 2               *                              ;
;               *************************                              ;
;                       ...                                            ;
;                       ...                                            ;
;                       ...                                            ;
;               *************************                              ;
;               * ENTRY N               *                              ;
;               *************************                              ;
;                                                                      ;
;   Note: On return, AX contains which entry was selected, BX points   ;
;   to buffer descriptor or LOCAL DATA address if descriptor next      ;
;   pointer is non-null.                                               ;
;                                                                      ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                                                      ;
; Task Control Block Layout                                            ;
;                                                                      ;
; TASK_ID:      each block is pointed to by a unique ID, which is the  ;
;               offset (address) within the DA code segment of the TCB ;
;               for that TASK.                                         ;
;                                                                      ;
;                                                                      ;
; Sync_Semaphore: The sync semaphore is used to suspend (or resume)    ;
;               execution of the associated task for rendezvous.       ;
;                                                                      ;
; Entry_Table: The Entry table provides a record for each of the       ;
;               entries defined in the task.  The record contains:     ;
;                       WAITING : flag indicating that the accepting   ;
;                                 task is waiting for an entry call    ;
;                                 for this entry.                      ;
;                       NEXT_PTR: Head of buffer descriptor linked to  ;
;                                 this entry.                          ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
        .LIST
```

```
     .XLIST
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;   FILE:  DA_HW.ASM                                               ;
;   Distributed Ada - Hardware Definition Include File             ;
;                                                                  ;
;;;;;;;;;;;;;;;;;;;;;; Distribution and Copyright ;;;;;;;;;;;;;;;;;;;
;   Derivation   : LabTek Distributed Ada V1.0                     ;
;                                                                  ;
;   This Distributed Ada Runtime inherits the LabTek copyright.    ;
;   The following copyright must be included in all software       ;
;   utilizing this Ada Runtime.                                    ;
;                                                                  ;
;   Copyright 1989 by LabTek Corporation, Woodbridge, CT, USA      ;
;                                                                  ;
;   Permission to use, copy, modify, and distribute this           ;
;   software and its documentation for any purpose and without     ;
;   fee is hereby granted, provided that the above copyright       ;
;   notice appear in all copies and that both that copyright       ;
;   notice and this permission notice appear in supporting         ;
;   documentation, and that the name of LabTek not be used in      ;
;   advertising or publicity pertaining to distribution of the     ;
;   software without specific, written prior permission.           ;
;   LabTek makes no representations about the suitability of       ;
;   this software for any purpose.  It is provided "as is"         ;
;   without express or implied warranty.                           ;
;                                                                  ;
;;;;;;;;;;;;;;;;;;;; Disclaimer ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                                                  ;
;   This software and its documentation are provided "AS IS" and   ;
;   without any expressed or implied warranties whatsoever.        ;
;   No warranties as to performance, merchantability, or fitness   ;
;   for a particular purpose exist.                                ;
;                                                                  ;
;   In no event shall any person or organization of people be      ;
;   held responsible for any direct, indirect, consequential       ;
;   or inconsequential damages or lost profits.                    ;
;                                                                  ;
;;;;;;;;;;;;;;;;;;;; END-PROLOGUE ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;; ;
;  Ethernet Board Hardware Configuration                           ;
;                                                                  ;
base                      equ     310H     ; base address of board
vector_number             equ       5H     ; vector number for board
net_memory_seg            equ    0DC00H    ; address of ethernet memory
net_memory_size           equ     2000H    ; 8K bytes


;
;   LAN Controller Page 0 registers
;

NIC_cr        equ  base + 0;          -- control register of NIC
```

```
NIC_pstart      equ  base + 1;          -- page start register
NIC_pstop       equ  base + 2;          -- page stop register
NIC_bndy        equ  base + 3;          -- boundary register
NIC_tpsr        equ  base + 4;          -- transmit page start register
NIC_tbcr0       equ  base + 5;          -- transmit byte count rgtr hi
NIC_tbcr1       equ  base + 6;          -- transmit byte count rgtr lo
NIC_isr         equ  base + 7;          -- interrupt status register
NIC_rsar0       equ  base + 8;          -- remote start address rgtr lo
NIC_rsar1       equ  base + 9;          -- remote start address rgtr hi
NIC_rbcr0       equ  base + 10;         -- remote byte count rgtr lo
NIC_rbcr1       equ  base + 11;         -- remote byte count rgtr hi
NIC_rcr         equ  base + 12;         -- receive configuration rgtr
NIC_tcr         equ  base + 13;         -- transmit configuration rgtr
NIC_dcr         equ  base + 14;         -- data configuration register
NIC_imr         equ  base + 15;         -- interrupt mask register
;
; controller page 1 registers - NIC address setup registers
; These registers are written to establish what the actual
; physical address will be.
;
phys_address_0 equ base + 1;      ; physical address registers.
phys_address_1 equ base + 2;      ; These registers are accessed
phys_address_2 equ base + 3;      ; via NIC_cr bits 7,6 = 0,1.
phys_address_3 equ base + 4;      ; LAN registers are accessed
phys_address_4 equ base + 5;      ; via cntrl bits 3,2 = 0,0.
phys_address_5 equ base + 6;      ;
NIC_curr       equ base + 7;      ; only written once during init


;
;   Controller Page 2 - Ethernet PROM ADDRESS memory
;   These locations contain the "preferred" address as contained
;   in PROM.  These will typically be copied to the physical
;   address registers above (page 1).
;
prom_address_0 equ  base + 0;           -- station address 0
prom_address_1 equ  base + 1;           -- station address 1
prom_address_2 equ  base + 2;           -- station address 2
prom_address_3 equ  base + 3;           -- station address 3
prom_address_4 equ  base + 4;           -- station address 4
prom_address_5 equ  base + 5;           -- station address 5


;
;   Gate Array registers (note: offset of 400H)
;

pstr            equ  base + 400H;    -- page start register
pspr            equ  base + 401H;    -- page stop register
dqtr            equ  base + 402H;    -- drq timer register
bcfr            equ  base + 403H;    -- base configuration register
pcfr            equ  base + 404H;    -- prom configuration register
gacfr           equ  base + 405H;    -- ga configuration register
```

```
cntrl          equ  base + 406H;     -- gate array (ga) control rgtr
streg          equ  base + 407H;     -- ga status register
idcfr          equ  base + 408H;     -- interrupt/DMA cnfgrtn rgtr
damsb          equ  base + 409H;     -- DMA address register hi
dalsb          equ  base + 40AH;     -- DMA address register lo
vptr2          equ  base + 40BH;     -- vector pointer rgtr #2
vptr1          equ  base + 40CH;     -- vector pointer rgtr #1
vptr0          equ  base + 40DH;     -- vector pointer rgtr #0
rfmsb          equ  base + 40EH;     -- register file access hi
rflsb          equ  base + 40FH;     -- register file access lo

;******************************************
;* Ethernet (3com) Initialization Values *
;******************************************

eth_enable_reset    equ   03h          ; enable reset
eth_disable_reset   equ   00h          ; disable reset
eth_access_prom     equ   04h          ; access prom bytes
eth_recv_select ,   equ   00h          ; select external Xceiver
eth_lan_config      equ   49h          ; 8k of mem-map I/O, w/interrupts
eth_rem_DMA_burst   equ   08h          ; # of bytes to transfer on DMA burst
eth_irq_line        equ   80h          ; interrupts occur on IRQ5
eth_rem_DMA_config  equ   20h          ; 8k configuration for remote DMA
eth_xmit_buf_start  equ   20h          ; begin of transmission buffer (0H)
eth_recv_buf_start  equ   26h          ; receive queue          (0600H)
eth_recv_buf_end    equ   40h          ; 20 pages, 256 bytes/page  (2000H)
eth_offset          equ   2000h        ; difference between page & address
eth_recv_begin      equ   600h         ; actual offset in RAM seg for begin
eth_recv_end        equ   2000h        ; actual offset in RAM seg for end
eth_start_nic       equ   02h          ; start NIC
eth_nic_stop        equ   01h          ; stop the NIC
eth_nic_DMA_config  equ   48h          ; local DMA operations, 8 byte bursts
eth_remote_DMA_lo   equ   00h          ; DMA remote unused (lo)
eth_remote_DMA_hi   equ   00h          ; DMA remote unused (hi)
eth_packet_types    equ   0fh          ; receive any kind of packet
eth_nic_mode        equ   02h          ; internal loopback mode
eth_bndy_start      equ   00h          ; FOR NOW, DO NOT USE BOUNDRY REG!
eth_int_status      equ   0ffh         ; clear status of all ints at start
eth_ints_enabled    equ   00h          ; enable no interrupts
eth_access_page_0   equ   00h          ; access page 0 again
eth_access_page_1   equ   40H          ; access NIC page 1 registers
eth_exit_mode       equ   00h          ; exit internal loopback mode

nic_prx             equ    1           ; mask for packet receive interrupt
nic_ptx             equ    2           ; mask for packet transmit interrupt

send                equ    4           ; command byte to start transmission

;
;  Interrupt Controller Command
;
```

```
NET_EOI         equ  60H + vector_number   ; -- End Of Interrupt (specific)

; Ethernet controller routine specifications
;
; Ethnet_Init initializes a 3com Etherlink II board to transmit and receive
; packets via a memory mapped interface with the board located at DC00:0000.
; The base address from which the registers are located is 310h.  The init
; routine intializes the memory to zeroes before it completes.  Although no
; DMA is used to transfer the data from main memory to the board's memory
; (which is referred as remote DMA operations), there is no choice but to
; use the local DMA operations (transferring bytes or words from the board's
; memory to the board's output fifo's).
      .LIST
```

```
        page    55,132
        TITLE   IO - Distributed Ada Network IO
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; FILE:  DA_IO.ASM                                                    ;
; IO  MODULE - Low Level Network Functions                            ;
;                                                                     ;
;;;;;;;;;;;;;;;;;;;;;; Distribution and Copyright ;;;;;;;;;;;;;;;;;;;;;
;  Derivation    : LabTek Distributed Ada V1.0                        ;
;                                                                     ;
;  This Distributed Ada Runtime inherits the LabTek copyright.        ;
;  The following copyright must be included in all software           ;
;  utilizing this Ada Runtime.                                        ;
;                                                                     ;
;  Copyright 1989 by LabTek Corporation, Woodbridge, CT, USA          ;
;                                                                     ;
;  Permission to use, copy, modify, and distribute this               ;
;  software and its documentation for any purpose and without         ;
;  fee is hereby granted, provided that the above copyright           ;
;  notice appear in all copies and that both that copyright           ;
;  notice and this permission notice appear in supporting             ;
;  documentation, and that the name of LabTek not be used in          ;
;  advertising or publicity pertaining to distribution of the         ;
;  software without specific, written prior permission.               ;
;  LabTek makes no representations about the suitability of           ;
;  this software for any purpose.  It is provided "as is"             ;
;  without express or implied warranty.                               ;
;                                                                     ;
;;;;;;;;;;;;;;;;;;;; Disclaimer ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                                                     ;
;  This software and its documentation are provided "AS IS" and      ;
;  without any expressed or implied warranties whatsoever.            ;
;  No warranties as to performance, merchantability, or fitness       ;
;  for a particular purpose exist.                                    ;
;                                                                     ;
;  In no event shall any person or organization of people be          ;
;  held responsible for any direct, indirect, consequential           ;
;  or inconsequential damages or lost profits.                        ;
;                                                                     ;
;;;;;;;;;;;;;;;;;;;; END-PROLOGUE ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                                                     ;
; The IO module provides the low_level interface to the network       ;
; hardware and receive message buffering.                             ;
;                                                                     ;
; This code is loaded into all processors, and adapts to the          ;
; the network hardware in its host.  Which routines are used is       ;
; determined solely by the calls made from the application code       ;
; and the messages received.                                          ;
;                                                                     ;
; The IO interface is implemented as four separate functions:         ;
;    Initialize                                                       ;
;    Transmit                                                    ;
;    Receive                                                     ;
;    Interrupt Procesing                                         ;
;                                                                ;
; The initialize function obviously must be called prior to any  ;
; other, and establishes the interrupt vector and enables, as    ;
; well as prepares the hardware for use.  It is also responsible ;
; for initilizing data structures used to buffer incoming packets. ;
;                                                                ;
; The Transmit function is used by one task at a time, and is    ;
; guarded by a semaphore to provide mutual exclusion.  Once the  ;
; transmit resource is granted, the data is copied into the on-card;
; buffer and sent out via hardware commands.  (Normally, hardware ;
; packet acknowledge should be provided and therefore it is not  ;
; implemented in software, even though Ethernet does not support ;
; hardware acknowledge.)                                         ;
;                                                                ;
; The Receive function is provided to assist in transferring the ;
; data to the requested destination.                             ;
;                                                                ;
; The Interrupt Processing handles both transmit complete and    ;
; reception interrupts.  For transmit complete, the resource is  ;
; simply made available again by performing a V operation on the ;
; trasmit semaphore.  For Receive interrupts, a buffer is allocated;
; from a linked list of fixed sized buffers.  Then the incoming  ;
; data is copied to the buffer and the distributed runtime is    ;
; invoked to process the request.  It may simply post the fact the ;
; message has arrived (and queue to an entry), or it may cause a ;
; task to resume which involves signalling (V - operation) the   ;
; suspended task.                                                ;
;                                                                ;
; Refer to individual procedure headers for parameter information ;
; and calling requirements.                                      ;
;                                                                ;
;                                                                ;
; Ver Date      Description                                      ;
;                                                                ;
; 0.1 Nov-88 : Initial prototype                                 ;
;                                                                ;
;                                                                ;
;                                                                ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

        .model  large
        include DA_DEF.ASM              ; contains software definitions
        include DA_HW.ASM               ; contains hardware specifics

        public  IO_Network_Init, IO_Xmit
        public  TX_READY                ; semaphore
        public  IO_ALLOCATE, IO_DEALLOCATE
        extrn   VRTIF_Signal_I:far       ; signal semaphore
        extrn   VRTIF_Wait:far           ; wait on semaphore "P"
        extrn   VRTIF_I8259:abs          ; address of 8259
        extrn   VRTIF_vector_base:abs    ; base of vector table
        extrn   Setup:near               ; Initialize Network I/F
        extrn   NET_Receive:near
;
;  software support buffers
;
buff_size               equ     2048    ; bytes in local buffer
num_buff                equ     20      ; number of buffers

cseg    segment common

        org     0C00H
        assume  cs:cseg,ds:cseg,es:cseg,ss:sseg

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;   NETWORK_INIT  : load Interrupt Vector and clear pointers  ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
IO_Network_Init:
        push    ax
        push    bx
        push    cx
        push    dx
        push    ds

;
;  Do low level Network Interface Card Initialization
;
        call    Setup
;
;  init network variables
;
        mov     ax,cs
        mov     ds,ax
        mov     [RECEIVE_PTR],eth_recv_begin   ; receive pointer
;
;  Initialize Receive buffer list
;
        lea     ax,RX_BUFF_Q
        mov     [RX_BUFF_HEAD],ax

        lea     ax,RX_BUFFER    ; points to actual buffers
        mov     cx,num_buff     ; number to link
        lea     bx,RX_BUFF_Q    ; points to buffer descriptors
Init30:
        mov     [bx],ax         ; put in current buffer pointer
        lea     dx,[bx+4]       ; DX is address of next descriptor
        mov     [bx+2],dx       ; put it in as next pointer
        add     ax,buff_size    ; point AX at next buffer
        mov     bx,dx           ; change descriptor pointer to next
        loop    Init30
;
;  now fix up last pointer
;
        mov     word ptr [bx-2],0        ; terminate list


;
;  load interrupt vector
;
        mov     ax,0
        mov     ds,ax
        mov     bx,VRTIF_vector_base+(vector_number*4)
        mov     ax,offset Interrupt_Handler
        mov     [bx],ax
        mov     ax,cs
        mov     [bx+2],ax
;
;  Note: Preliminary board initialization was done in SETUP code, now
;        just enable interrupts
;
        mov     dx,VRTIF_I8259+1
        in      al,dx                   ; get interrupt mask
        mov     ah,0FEH                 ; mask to clear zero bit
        mov     cl,vector_number        ; load shift count register
        rol     ah,cl
        and     al,ah                   ; enable level
        out     dx,al                   ; update controller chip

        mov     dx,cntrl
        mov     al,eth_access_page_1    ; access NIC page 1 registers
        out     dx,al

        mov     dx,nic_imr              ; interrupt mask register
        mov     al,nic_prx+nic_ptx      ; enable xmit/recv interrupts
        out     dx,al

        pop     ds
        pop     dx
        pop     cx
        pop     bx
        pop     ax
        ret


header_size     equ     10 ;words:dst=3,src=3,RCP=1,priority=1,seq=1,length=1

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;  XMIT -  transmit the message specified by parameter list       ;
;          starting at address is at SS:bp+DEF_TCP                ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

```
;  INPUTS:      TRANSMIT CONTROL POINTER
;               Number of parameters
;               offset1
;               segment1
;               length1
;                      ... etc.
IO_Xmit:
        push    cs                    ; push segment of transmit ctrl semaphore
        lea     ax,TX_READY
        push    ax                    ; push offset of semaphore
        call    VRTIF_Wait            ; do p semaphore operation


;
;  put header in packet buffer
;
        mov     si,[bp+DEF_TCP]       ; fetch Transmit control pointer
        les     di,[CARD_RAM]         ; point to hardware buffer area
        mov     cx,header_size        ; in words
        mov     [PACKET_SIZE],cx      ; initialize packet size (in words)
        sub     cx,2                  ; do not copy sequence and length fields
        pushf
        cli
        repz    movsw
        popf
;
;  Update Sequence Number and put it in packet
;
        mov     si,[si]           ; fetch sequence offset
        mov     ax,[si]           ; get sequence count
        inc     ax
        stosw                     ; put sequence count in packet
        mov     [si],ax           ; update counter  ;
;  skip over length field for now
;
        add     di,2
;
;  copy the parameters into the packet buffer
;
        mov     si,DEF_Parameters         ; offset from BP to parameter list
        mov     cx,[bp+si+DEF_param_count] ; number of parameters
        jcxz    Xmit_20                   ; if no parameters
        add     si,2                      ; increment to actual parameter info
Xmit_10:
        push    cx
        mov     cx,[bp+si+DEF_param_length]; get size of parameter (bytes)
        shr     cx,1                      ; convert to words
        add     [PACKET_SIZE],cx          ; accumulate into packet size counter
        push    ds
        lds     si,[bp+si+DEF_PARAM_OFFSET]; get address of parameter
        rep     movsw                     ; copy data to packet buffer
        pop     ds
        add     si,DEF_param_descriptor ; go to next parameter descriptor
        pop     cx                      ; get parameter count back
        loop    Xmit_10
;
; Now all parameters have been copied in, now insert length field
;
Xmit_20:
        les     di,[CARD_RAM]           ; point to hardware buffer area
        add     di,data_length          ; add offset to data length field
        mov     ax,[PACKET_SIZE]
        stosw                           ; stick in PACKET length
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Setup NIC registers to begin transmission                                   ;
; Must prevent a RECEIVE interrupt from arriving, which would interfere       ;
; with the registers being updated for Transmission.                          ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;  ;
; load start address of packet
;
        pushf                           ; save interrupt status
        cli                             ; disable any interrupts
        mov     dx,nic_cr               ; select Page_0
        mov     al,eth_access_Page_0
        out     dx,al

        mov     dx,nic_tpsr             ; page start register
        mov     al,eth_xmit_buf_start   ; transmit page at DC00:0000
        out     dx,al
; load length of packet
        mov     ax,[PACKET_SIZE]
        shl     ax,1                    ; convert word to byte count
        mov     dx,nic_tbcr0
        out     dx,al
        mov     dx,nic_tbcr1
        mov     al,ah
        out     dx,al
; start transmit
        mov     dx,nic_cr
        mov     al,send         ; command to initiate transmission
        out     dx,al
        popf                    ; restore interrupt status
        ret

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
;  INTERRUPT SERVICE ROUTINE
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
; Currently, this must have a stack frame similar to other vendor
; interrupt routines so that the interrupt-mode Signal routine will
; be able to find the interrupt return address and status
```

```
;
Interrupt_Handler       label   far
        push    bp
        mov     bp,sp
        push    ax
        push    bx
        push    cx
        push    dx
        push    si
        push    di
        push    ds
        push    es

;  Setup data segment

        mov     ax,seg cseg
        mov     ds,ax
;
;  First fetch interrupt status, and clear interrupt request
;
        mov     dx,nic_cr       ; select Page_0
        mov     al,eth_access_Page_0
        out     dx,al

        mov     dx,nic_isr      ; look at interrupt status register
        in      al,dx
        mov     [ISR],ax        ; save status
        out     dx,al           ; clear interrupts
;
;  Clear the 8259 Interrupt Request
;
        mov     al,NET_EOI      ; issue EOI to interrupt controller
        mov     dx,VRTIF_I8259
        out     dx,al


;
;  check for receive interrupt
;
        test    [ISR],nic_prx   ; see if receive interrupt
        jz      Check_Xmit
;
;  Receive complete interrupt, process incoming packet
;  NOTE: since this is done inside the interrupt routine, interrupts
;  are disabled, and therefore there is no interference from possible
;  receive interrupts.
;
Receive:
;
;  Allocate a buffer, and transfer data to the buffer
;  after the following call, the buffer descriptor is in BX. DO NOT DESTROY BX!
;
```

```
        call    IO_Allocate
        mov     ax,cs           ; destination segment is CS
        mov     es,ax
        mov     di,[bx]         ; destination offset is buffer at BX
        lds     si,[CARD_RAM]   ; source is ethernet RAM
        add     si,cs:[RECEIVE_PTR]; add current receive buffer page address
        lodsw                   ; fetch status into AL, NEXT PTR into AH
        xor     al,al           ; zero low byte, leaving a new pointer
        sub     ax,eth_offset   ; correct for memory vs page offset
        mov     cs:[RECEIVE_PTR],ax; get ready for next reception
        add     si,2            ; skip over receive byte count
;  SI now points to first part of transmitted packet
        mov     ax,[si+data_length] ; get size of valid packet in WORDs
;
;  Now transfer memory from hardware buffer pages to software buffer.
;  Note that the buffer will wrap around at 4000H back to 2600.
;
        mov     dx,80H-2        ; page size in words (reduced to get aligned)
RECV010:
        cmp     ax,dx           ; see if more than a page
        jge     RECV020
        mov     dx,ax           ; otherwize only move the remaining
RECV020:
        mov     cx,dx
        rep     movsw           ; do the transfer
        cmp     si,eth_recv_end ; see if at end of hardware buffer
        jnz     RECV030
        mov     si,eth_recv_begin; reset pointer to begin
RECV030:
        sub     ax,dx           ; reduce total count by those moved
        jz      RECV040         ; finished if so
        mov     dx,80H          ; keep page alignment
        jmp     RECV010
RECV040:
        mov     ax,cs           ; restore data segment
        mov     ds,ax
;
;  Call Receive portion of Distributed Runtime code to determine
;  what should be done with the newly arrived packet.
;
        call    NET_Receive
;
;   Check to see if any more work to do, if so then skip clearing
;   the interrupt so that another interrupt will occur imediately
;   upon enabling interrupts (possibly after a trip through the
;   scheduler).
;    Note that a limitation in the Ethernet hardware results in
;    a possible race condition here.

        cli                     ; reduce chance of race (in case VRT enabled)
        mov     dx,nic_cr
        mov     al,eth_access_Page_1 ; select page 1
        out     dx,al

        mov     dx,nic_curr     ; get current page register
        in      al,dx           ; fetch current page register
        mov     ah,al           ; build memory address
        xor     al,al
        sub     ax,eth_offset   ; correct for NIC displacement
        cmp     ax,[RECEIVE_PTR]; check if our pointer is the same
        jz      Check_Xmit      ; if no work, don't print notice

                                ; this will result in an immediate re-interrupt
                                ; as soon as interrupts are enabled
                                ; (which means after scheduling event)
;
;  Now check for transmit complete interrupt
;
Check_Xmit:

        test    [ISR],nic_ptx   ; check for packet transmitted
        jz      EOI
;
;  Transmit complete, first clear interrupt request, then signal semaphore
;
Transmit:
        push    cs              ; segment of semaphore
        lea     ax,TX_READY     ; offset of semaphore
        push    ax
        call    VRTIF_Signal_I  ; signal completion

EOI:
        pop     es
        pop     ds
        pop     di
        pop     si
        pop     dx
        pop     cx
        pop     bx
        pop     ax
        pop     bp
        iret


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;  IO_ALLOCATE - Allocates next buffer from Avail list                    ;
;       Return BX pointing to buffer queue index.                         ;
;       By design, the buffer should queue should never be empty.         ;
;  Destroys AX  , BX has new descriptor pointer                           ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
IO_ALLOCATE:
        pushf
        cli
        mov     bx,[RX_BUFF_HEAD]       ; fetch head pointer
        or      bx,bx                   ; see if empty
        jnz     IO_ALLOC10              ; go on if not
;
; Normally, might raise storage error here, but design prevents
; exceeding buffer capacity unless there is some code flaw.
;
        popf
        int     3                       ; trap
;
; Remove buffer descriptor from free list
;
IO_ALLOC10:
        mov     ax,[bx+DEF_NEXT_PTR]    ; fetch next pointer
        mov     [RX_BUFF_HEAD],ax       ; pull buffer off list, replace head
        xor     ax,ax                   ; null next pointer in buffer
        mov     [bx+DEF_NEXT_PTR],ax
        popf
        ret

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; IO_DEALLOCATE - Deallocates buffer into Avail list                       ;
;       Takes BX pointing to buffer descriptor.                            ;
;      By design, the buffer should queue should never be full.            ;
; Destroys AX                                                              ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
IO_DEALLOCATE:
        pushf
        cli
        mov     ax,[RX_BUFF_HEAD]       ; get head of list
        mov     [bx+DEF_NEXT_PTR],ax    ; put behind this entry
        mov     [RX_BUFF_HEAD],bx       ; make this entry new head
        popf
        ret


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Data AREA
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
        align   4
ISR             dw      0               ; interrupt status register
PACKET_SIZE     dw      0               ; packet size

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; BUFFER QUEUE STRUCTURE          ;
;       record                    ;
;         BUFFER_OFFSET           ;
;         NEXT_PTR                ;
;       end record;               ;
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

CARD_RAM        dd      0dc000000h          ; address of ram buffer on enet card
RECEIVE_PTR     dw      ?                   ; points to current next page to rcv
;
;  The following semaphore is used to provide mutual exclusion to the
;  transmit side of the Ethernet card.
;
TX_READY        dw      1                 ; semaphore count
                dw      0                 ; task value
                dw      0                 ; task value

RX_BUFF_HEAD    dw      (?)
RX_BUFFER       db      num_buff dup (buff_size dup (?))
RX_BUFF_Q       dw      num_buff dup (2 dup(?))  ; (BUFFER_PTR, NEXT_DESC_PTR)


cseg      ends


sseg    segment STACK
        dw      100 dup (0)
sseg    ends

        end
        page    55,132
        TITLE   LINK - Distributed Ada Linkage Module
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;  FILE:  DA_LINK.ASM                                                        ;
;                                                                            ;
;  LINK - Distributed Ada - Linkage Module                                   ;
;                                                                            ;
;;;;;;;;;;;;;;;;;;;;;; Distribution and Copyright ;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;   Derivation    : LabTek Distributed Ada V1.0                              ;
;                                                                            ;
;   This Distributed Ada Runtime inherits the LabTek copyright.              ;
;   The following copyright must be included in all software                 ;
;   utilizing this Ada Runtime.                                              ;
;                                                                            ;
;   Copyright 1989 by LabTek Corporation, Woodbridge, CT, USA                ;
;                                                                            ;
;   Permission to use, copy, modify, and distribute this                     ;
;   software and its documentation for any purpose and without               ;
;   fee is hereby granted, provided that the above copyright                 ;
;   notice appear in all copies and that both that copyright                 ;
;   notice and this permission notice appear in supporting                   ;
;   documentation, and that the name of LabTek not be used in                ;
;   advertising or publicity pertaining to distribution of the               ;
;   software without specific, written prior permission.                     ;
;   LabTek makes no representations about the suitability of                 ;
;   this software for any purpose.  It is provided "as is"                   ;
;   without express or implied warranty.                                     ;
;                                                                            ;
;;;;;;;;;;;;;;;;;;;; Disclaimer ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                                                            ;
;   This software and its documentation are provided "AS IS" and            ;
;   without any expressed or implied warranties whatsoever.                  ;
;   No warranties as to performance, merchantability, or fitness             ;
;   for a particular purpose exist.                                          ;
;                                                                            ;
;   In no event shall any person or organization of people be               ;
;   held responsible for any direct, indirect, consequential                 ;
;   or inconsequential damages or lost profits.                              ;
;                                                                            ;
;;;;;;;;;;;;;;;;;;;; END-PROLOGUE ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                                                            ;
;  This code is code that would be generated by the Compiler/Linker ;
;  and Distributor.  It is kept separate from the runtime "routines";
;  to delineate the generated/runtime boundry.                              ;
;                                                                            ;
;  This code is Linked to the Ada application via (hand) editing            ;
;  of runtime calls within the generated Ada program.  Essentially, ;
;  a call is made from each of the rendezvous operations in the             ;
;  generated call to the respective support code here, then a return;
;  is made back.  Since the compiler does not supply information            ;
; on the parameters in the code (it is implicitly maintained by   ;
; the compiler amoung entry call/accept pairs), a little support  ;
; code is placed here to provide the information.  Normally a     ;
; compiler that supports distribution would put this in line.     ;
;                                                                 ;
; The support code then calls general purpose (although prototype) ;
; routines to implement remote entry, accept, select, and error   ;
; mechanisms.                                                     ;
;                                                                 ;
; Each packet header is statically formed and placed in this      ;
; module to be reference by the TRANSMIT CONTROL PTR (TCP) used in ;
; the runtime call parameter list.  This reduces the overhead     ;
; associated with packetizing the data.  These packet headers     ;
; could be generated by the compiler/linker/distributor and       ;
; optimally would be placed in the controller card memory at      ;
; elaboration time so that loading of header data would be        ;
; necessary.                                                      ;
;                                                                 ;
;                                                                 ;
; Ver Date      Description                                       ;
;                                                                 ;
; 0.1 Nov-88 : Initial prototype                                  ;
;                                                                 ;
;                                                                 ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

        include DA_DEF.ASM

        .model  large

        extrn   Initialize:far

        extrn   remote_entry:far, local_entry:far
        extrn   Any_select:far, remote_accept:far, remote_end_accept:far
        extrn   local_end_accept:far
        extrn   remote_elab_start:far, remote_elab_wait:far
        extrn   remote_elab_continue:far
        extrn   IO_DEALLOCATE:near
        extrn   outchr:near

cseg    segment common


;
;  Lengths of rendezvous data parameters
;
ROCKET_MSG_TYPE_LENGTH          equ     202
ROCKET_GUIDE_MSG_TYPE_LENGTH    equ     122
TARGET_MSG_TYPE_LENGTH          equ     1002
;
;  Rendezvous parameter offsets
```

```
;
NEXT_ROCKET_MSG             equ     1908    ; offset from BP
CONTROL_GUIDE_MSG           equ      386    ; offset from BP

ROCKSUP_GUIDE_MSG           equ     -334    ; offset from BP
ROCKSUP_REPORT_MSG          equ     -212    ; offset from BP

TARGET_MSG                  equ    -1012    ; offset from BP

        assume  cs:cseg,ds:cseg,es:cseg

;
; The following jump table provides (static) control transfers from the
; Ada application code to the respective support code located here
;
        align  8
  jmp   Init                    ; prior to elaboration

        align  8
  jmp   Elaborate_Start         ; to synchronize elaboration
        align  8
  jmp   Elaborate_Wait          ; will wait for elaborate sync.
        align  8
  jmp   Elaborate_Continue      ; to acknowledge completion

        align  8
  jmp   get_report_entry        ; called by Rocket.Control
        align  8
  jmp   put_report_entry        ; called by Simulate.RDL.Rocket_Support
        align   8
  jmp   report_buf_select       ; called by Simulate.RDL.Report_Buf
        align  8
  jmp   get_report_end_accept   ; called by Simulate.RDL.Report_Buf
        align   8
  jmp   put_report_end_accept   ; called by Simulate.RDL.Report_Buf

        align  8
  jmp   get_guide_entry         ; called by Simulate.RDL.Rocket_Support
        align  8
  jmp   guide_buf_select        ; called by Simulate.RDL.Guide_Buf
        align  8
  jmp   put_guide_entry         ; called by Rocket.Control

        align  8
  jmp   get_next_target_entry   ; called by Target.Track
        align  8
  jmp   get_next_target_accept   ; called by Simulate.Sensor.Targ_Support
        align   8
  jmp   get_next_target_end_accept; called by Simulate.Sensor.Targ_Support
        align   8
  jmp   get_guide_end_accept    ; called by Simulate.RDL.Guidebuf
```

```
        align   8
  jmp   put_guide_end_accept    ; called by Simulate.RDL.Guidebuf
        align  80H
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
;   INIT  to initialize the network hardware
;
Init:
        push    ds
        mov     ax,cs
        mov     ds,ax
        call    Initialize
        pop     ds
        retf

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
;   ELABORATE_START
;   The main program calls this routine to send a "start" to the
;   bravo processor.  The remote_elab_start routine will suspend
;   the main program until the continue is received from the
;   elaborating processor.
;
Elaborate_Start:
        push    ds
        mov     ax,cs
        mov     ds,ax
        xor     ax,ax                   ; no parameters
        push    ax
        lea     ax,TX_elaborate_begin   ; transmit control ptr
        push    ax
        lea     ax,Main_TCB             ; task_id
        push    ax
        call    remote_elab_start
        add     sp,6                    ; fix up stack
;
;   Normally, test AX for NZ to see if there was an elaboration
;   error on the remote processor, however it will be ignored
;   in this case.
;
        pop     ds
        retf

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
;   ELABORATE_WAIT : Suspends a task until it is given the signal
;                    to continue from the elaborating processor.
;
Elaborate_Wait:
        push    ds
        mov     ax,cs
        mov     ds,ax
        lea     ax,Main_TCB.ENTRY1_Q     ; put main entry queue (pseudo)
        push    ax
        lea     ax,Main_TCB
        push    ax              ; put main TCB as task_id
        call    remote_elab_wait
        add     sp,4
        pop     ds
        retf

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
;   ELABORATE_CONTINUE : Notifies the elaborating processor that
;                        the specified elaboration in complete and
;                        that it can continue elaboration of other
;                        units.
;
Elaborate_Continue:
        push    ds
        mov     ax,cs
        mov     ds,ax
        xor     ax,ax           ; no parameters
        push    ax
        lea     ax,Main_TCB.ENTRY1_Q     ; pass entry queue in place of TCP
        push    ax                       ; RTE will deallocate incoming buffer
        lea     ax,Main_TCB     ; and calling task id
        push    ax
        call    remote_elab_continue
        add     sp,6            ; remove parameters
        pop     ds
        retf

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;   GET_REPORT_ENTRY
;
get_report_entry:         ; called by Rocket.Control
        push    ds
        mov     ax,cs
        mov     ds,ax
        mov     al,'R'
        call    outchr
        mov     al,'G'
        call    outchr
        mov     al,' '
        call    outchr
;
;  Push parameters for remote rendezvous entry
;
        xor     ax,ax     ; no IN parameters, set count to zero
        push    ax
        lea     ax,TX_report_begin_entry        ; transmit control ptr
        push    ax
        lea     ax,Control_TCB  ; rocket.control task
        push    ax              ; push task_id
        call    remote_entry
        add     sp,6            ; clean up stack
        mov     bx,word ptr [Control_TCB+DEF_TCB_REPLY]  ; get reply pointer
;
;  Copy out parameter, (BX) points to buffer descriptor
;
        lea     di,[bp+NEXT_ROCKET_MSG]     ; point to NEXT_ROCKET_MSG
        mov     si,[bx]                     ; get address of buffer
        lea     si,[si+data_field]          ; get pointer to data in packet buff
        mov     cx,ROCKET_MSG_TYPE_LENGTH
        shr     cx,1                        ; word count
        mov     ax,ss
        mov     es,ax
        pushf
        cli
        rep     movsw
        popf
        call    IO_Deallocate        ;   Deallocate receive buffer
        pop     ds
        retf


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; NOTE: STACK FRAME LOCATIONS ARE POSITION DEPENDENT
;
report_buf_select:      ; called by Simulate.RDL.Report_Buf
        push    ds
        mov     ax,cs
        mov     ds,ax
        mov     al,'R'
        call    outchr
        mov     al,'s'
        call    outchr
        mov     al,' '
        call    outchr

        lea     ax,Reportbuf_TCB.ENTRY1_Q       ; get report
        push    ax
        lea     ax,Reportbuf_TCB.ENTRY2_Q       ; put report
        push    ax
        mov     ax,2                    ; number of entries (fixed)
        push    ax
        lea     ax,Reportbuf_TCB        ; task_id
        push    ax
        call    Any_select              ; parameter pointer & selector on stack
        add     sp,8                    ; restore stack
;
```

```
;   place return parameters on stack
;
        push    bp
        mov     bp,sp
        mov     [bp+8],ax               ; put selector on stack
        mov     [bp+10],bx              ; put offset of data pointer on stack
        mov     ax,cs
        mov     [bp+12],ax              ; put segment of data pointer on stack
        pop     bp

        pop     ds
        retf

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
get_report_end_accept:          ; called by Simulate.RDL.Report_Buf
        push    ds
        mov     ax,cs
        mov     ds,ax
        mov     al,'e'
        call    outchr
        mov     al,'A'
        call    outchr
        mov     al,' '
        call    outchr
        mov     ax,ROCKET_MSG_TYPE_LENGTH       ; length
        push    ax
;  put address of data buffer on stack (first and only parameter)
        push    word ptr es:[bx]                ; segment
        push    word ptr es:[bx+2]              ; offset
        mov     ax,1                            ; number of out parameters
        push    ax
        lea     ax,Reportbuf_TCB.ENTRY1_Q       ; entry_ID
        push    ax
        lea     ax,Reportbuf_TCB                ; task id
        push    ax
        call    remote_end_accept
        add     sp,12                           ; remove parameters from stack
        pop     ds
        retf

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
put_report_end_accept:          ; called by Simulate.RDL.Report_Buf
        push    ds
        mov     ax,cs
        mov     ds,ax
        mov     al,'E'
        call    outchr
        mov     al,'A'
        call    outchr
        mov     al,' '
        call    outchr
        lea     ax,Reportbuf_TCB.ENTRY2_Q
        push    ax
        call    local_end_accept
        add     sp,2            ; remove parameter
        pop     ds
        retf

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
;   NOTE: THIS IS A ENTRY CALL to a LOCAL TASK!
;
put_report_entry:       ; called by Simulate.RDL.Rocket_Support
        push    ds
        mov     ax,cs
        mov     ds,ax
        mov     al,'R'
        call    outchr
        mov     al,'P'
        call    outchr
        mov     al,' '
        call    outchr

        push    ss                              ; segment of data area
        lea     ax,[bp+ROCKSUP_REPORT_MSG]      ; offset of data
        push    ax
        lea     ax,Reportbuf_TCB.ENTRY2_Q       ; dst entry ID
        push    ax
        lea     ax,Reportbuf_TCB                ; dst task id
        push    ax
        lea     ax,Rocksup_TCB                  ; src task id
        push    ax
        call    local_entry
        add     sp,10                           ; restore stack
        pop     ds
        retf

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
get_guide_entry:        ; called by Simulate.RDL.Rocket_Support
        push    ds
        mov     ax,cs
        mov     ds,ax
        mov     al,'G'
        call    outchr
        mov     al,'E'
        call    outchr
        mov     al,' '
        call    outchr

        push    ss              ; segment of data area
        lea     ax,[bp+ROCKSUP_GUIDE_MSG] ; offset of data
        push    ax
        lea     ax,Guidebuf_TCB.ENTRY1_Q        ; dst entry ID
        push    ax
        lea     ax,Guidebuf_TCB                 ; dst task id
        push    ax
        lea     ax,Rocksup_TCB                  ; src task id
        push    ax
        call    local_entry
        add     sp,10                           ; restore stack
        pop     ds
        retf

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; NOTE: STACK FRAME LOCATIONS ARE POSITION DEPENDENT
guide_buf_select:       ; called by Simulate.RDL.Guide_Buf
        push    ds
        mov     ax,cs
        mov     ds,ax
        mov     al,'G'
        call    outchr
        mov     al,'S'
        call    outchr
        mov     al,' '
        call    outchr
        mov     ax,1                            ; always 1 entry (put_next)
        mov     cx,[bp-134]                     ; check MSG_COUNT
        cmp     cx,0
        jle     gbs10                           ; if guard closed
        inc     ax                              ; allow two entries
        lea     bx,Guidebuf_TCB.ENTRY1_Q        ; Get_next_guide
        push    bx
; put_next_guide is always open
gbs10:
        lea     bx,Guidebuf_TCB.ENTRY2_Q        ; Put_next_guide
        push    bx
        push    ax                      ; number of entries
        lea     ax,Guidebuf_TCB         ; task_id
        push    ax
        call    Any_select              ; parameter pointer & selector on stack
        pop     cx                      ; remove task_id
        pop     cx                      ; get number of entries
        shl     cx,1                    ; two bytes per entry
        add     sp,cx                   ; remove local stack frame
;
;   place return parameters on stack
;
        push    bp
        mov     bp,sp
        mov     [bp+8],ax               ; put selector on stack
        mov     [bp+10],bx              ; put offset of data pointer on stack
        mov     ax,cs
        mov     [bp+12],ax              ; put segment of data pointer on stack
        pop     bp

        pop     ds
        retf

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
get_guide_end_accept:           ; called by Simulate.RDL.Guide_Buf
        push    ds
        mov     ax,cs
        mov     ds,ax
        mov     al,'G'
        call    outchr
        mov     al,'e'
        call    outchr
        mov     al,' '
        call    outchr
        lea     ax,Guidebuf_TCB.ENTRY1_Q
        push    ax
        call    Local_End_Accept
        add     sp,2            ; remove parameter
        pop     ds
        retf

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
put_guide_end_accept:           ; called by Simulate.RDL.Guide_Buf
        push    ds
        mov     ax,cs
        mov     ds,ax
        mov     al,'G'
        call    outchr
        mov     al,'p'
        call    outchr
        mov     al,' '
        call    outchr
        mov     ax,0                            ; number of out parameters
        push    ax
        lea     ax,Guidebuf_TCB.ENTRY2_Q        ; entry_ID
        push    ax
        push    ax                              ; leave space normally task_ id
        call    remote_end_accept
        add     sp,6                            ; remove parameters from stack
        pop     ds
        retf

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
put_guide_entry:        ; called by Rocket.Control
        push    ds
        mov     ax,cs
        mov     ds,ax
        mov     al,'G'
        call    outchr
        mov     al,'P'
        call    outchr
        mov     al,' '
        call    outchr
        mov     ax,ROCKET_GUIDE_MSG_TYPE_LENGTH        ; length
        push    ax
        push    ss                                     ; current stack segment
        lea     ax,[bp+CONTROL_GUIDE_MSG]              ; offset
        push    ax
        mov     ax,1                                   ; 1 in parameter
        push    ax
        lea     ax,TX_guide_begin_entry                ; tx control pointer
        push    ax
        lea     ax,Control_TCB                         ; task id
        push    ax
        call    remote_entry
        add     sp,12                                  ; restore stack
        mov     bx,word ptr [Control_TCB+DEF_TCB_REPLY] ; get reply pointer
        call    IO_Deallocate                          ; no OUT entry params
        pop     ds
        retf

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
get_next_target_entry:   ; called by Target.Track
        push    ds
        mov     ax,cs
        mov     ds,ax
        mov     al,'T'
        call    outchr
        mov     al,'G'
        call    outchr
        mov     al,' '
        call    outchr

        xor     ax,ax                   ; no in parameters
        push    ax
        lea     ax,TX_track_begin_entry ; transmit control ptr
        push    ax
        lea     ax,Track_TCB            ; task_id
        push    ax
        call    remote_entry
        add     sp,6                    ; clean up stack
        mov     bx,word ptr [Track_TCB+DEF_TCB_REPLY]  ; get reply pointer
;
;   Copy out parameter, (BX) points to buffer descriptor, DI to control block
;
        lea     di,[bp+TARGET_MSG]     ; point to TARGET_MSG
        mov     si,[bx]                ; get actual buffer address
        lea     si,[si+data_field]     ; point to packet data
        mov     cx,TARGET_MSG_TYPE_LENGTH
        shr     cx,1                    ; make word count
        mov     ax,ss
        mov     es,ax
        pushf
        cli
        rep     movsw
        popf
        call    IO_Deallocate           ;  Deallocate receive buffer
        pop     ds
        retf

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
get_next_target_accept:   ; called by Simulate.Sensor.Targ_Support
        push    ds
        mov     ax,cs
        mov     ds,ax
        mov     al,'T'
        call    outchr
        mov     al,'a'
        call    outchr
        mov     al,' '
        call    outchr

        lea     ax,Targsup_TCB.ENTRY1_Q ;Entry Queue of interest
        push    ax
        lea     ax,Targsup_TCB          : our task_id
        push    ax
        call    remote_accept           ; returns: ES:BX-data ptr
        add     sp,4                    ; adjust stack back
        pop     ds
        retf

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
get_next_target_end_accept:   ; called by Simulate.Sensor.Targ_Support
        push    ds
        mov     ax,cs
        mov     ds,ax
        mov     al,'T'
        call    outchr
        mov     al,'b'
        call    outchr
        mov     al,' '
        call    outchr
        mov     ax,TARGET_MSG_TYPE_LENGTH
        push    ax
;  put address of data buffer on stack (first and only parameter)
        push    word ptr es:[bx]                ; segment
        push    word ptr es:[bx+2]              ; offset
        mov     ax,1                            ; number of out parameters
        push    ax
        lea     ax,Targsup_TCB.ENTRY1_Q         ; entry_ID
        push    ax
        push    ax                              ; leave space normally task_ id
        call    remote_end_accept
        add     sp,12                           ; remove parameters from stack
        pop     ds
        retf


        align   80H             ; keep addresses from changing

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;  Packet Header Data, pointed to by the TRANSMIT CONTROL_PTR  (TCP)    ;
;                                                                       ;
;  The following data structures are used to generate and respond to   ;
;  data packets.                                                        ;
;                                                                       ;
;                                                                       ;
;  type TRANSMIT_CONTROL_TYPE is record                                 ;
;    DESTINATION    : NET_ADDR;                                         ;
;    SOURCE         : NET_ADDR;                                         ;
;    RCP            : RECEIVE_CONTROL_PTR; -- Receive Control Field     ;
;    PRIORITY       : PRIORITY_TYPE;                                    ;
;    SEQUENCE_PTR   : SEQUENCE_TYPE;   -- offset to sequence counter    ;
;  end record;                                                          ;
;                                                                       ;
;  type REPLY_MODE_TYPE is (NO_REPLY, REPLY);                           ;
;                                                                       ;
;  type RECEIVE_CONTROL_TYPE(REPLY_MODE : REPLY_MODE_TYPE) is record    ;
;    COMMAND        : WORD range 0..8;    -- see command table below    ;
;    PROC_ID        : WORD;               -- processor ID               ;
;    TASK_ID        : WORD;                                             ;
;    ENTRY_ID       : WORD;                                             ;
;    case REPLY is                                                      ;
;      when TRUE =>                                                     ;
;        TX_CONTROL : TRANSMIT_CONTROL_TYPE;                            ;
;      when FALSE =>                                                    ;
;        null;                                                          ;
;    end case;                                                          ;
;  end record;                                                          ;
;                                                                       ;
;                                                                       ;

TX_elaborate_begin:
        DEF_bravo_address <>
        DEF_alpha_address <>
        dw      OFFSET RX_elaborate_begin       ; RECEIVE CONTROL
        dw      0                               ; priority
        dw      OFFSET BRAVO_SEQUENCER          ; sequence pointer

RX_elaborate_begin:
        dw      DEF_elab_begin                  ; elaborate begin COMMAND
        dw      DEF_pid_bravo                   ; dest. Proc ID (bravo)
        dw      offset Main_TCB                 ; dest. Task ID (Main)
        dw      offset Main_TCB.ENTRY1_Q        ; entry = elaborate
   ; reply Transmit control record
        DEF_alpha_address <>
        DEF_bravo_address <>
        dw      OFFSET RX_elaborate_end         ; RCP
        dw      0                               ; priority
        dw      OFFSET ALPHA_SEQUENCER          ; sequence pointer

RX_elaborate_end:
        dw      DEF_elab_end                    ; COMMAND = Elaborate End
        dw      DEF_pid_alpha                   ; dest. Proc ID (alpha)
        dw      offset Main_TCB                 ; dest. Task ID (MAIN)
        dw      offset Main_TCB.ENTRY1_Q

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

TX_report_begin_entry:
        DEF_bravo_address <>
        DEF_alpha_address <>
        dw      OFFSET RX_report_begin_entry    ; COMMAND = begin entry
        dw      0                               ; priority
        dw      OFFSET BRAVO_SEQUENCER          ; sequence pointer

RX_report_begin_entry:
        dw      DEF_request_entry               ; COMMAND = begin entry
        dw      DEF_pid_bravo                   ; dest. Proc ID (bravo)
        dw      offset Reportbuf_TCB            ; dest. Task ID (Report_Buf)
        dw      offset Reportbuf_TCB.ENTRY1_Q   ; entry = GET_NEXT_REPORT
   ; reply
        DEF_alpha_address <>
        DEF_bravo_address <>
        dw      OFFSET RX_report_end_entry      ; COMMAND = accept complete
        dw      0                               ; priority
        dw      OFFSET BRAVO_SEQUENCER          ; sequence pointer

RX_report_end_entry:
        dw      DEF_accept_complete             ; COMMAND = accept complete
        dw      DEF_pid_alpha                   ; dest. Proc ID (alpha)
        dw      offset Control_TCB              ; dest. Task ID (Control)
                                                ; caller entry is not required
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

TX_guide_begin_entry:
        DEF_bravo_address <>
        DEF_alpha_address <>
        dw      OFFSET RX_guide_begin_entry     ; COMMAND = begin entry
        dw      0                               ; priority
        dw      OFFSET BRAVO_SEQUENCER          ; sequence pointer
```

```
RX_guide_begin_entry:
        dw      DEF_request_entry               ; COMMAND = begin entry
        dw      DEF_pid_bravo                   ; dest. Proc ID (bravo)
        dw      offset Guidebuf_TCB             ; dest. Task ID (Guide_Buf)
        dw      offset Guidebuf_TCB.ENTRY2_Q    ; entry = PUT_NEXT_GUIDE
    ; reply
        DEF_alpha_address <>
        DEF_bravo_address <>
        dw      OFFSET RX_guide_end_entry       ; RCP COMMAND = accept complete
        dw      0                               ; priority
        dw      OFFSET ALPHA_SEQUENCER          ; sequence pointer


RX_guide_end_entry:
        dw      DEF_accept_complete             ; COMMAND = accept complete
        dw      DEF_pid_alpha                   ; dest. Proc ID (alpha)
        dw      offset Control_TCB              ; dest. Task ID (Control)
                                                ; caller entry not required
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

TX_track_begin_entry:
        DEF_bravo_address <>
        DEF_alpha_address <>
        dw      OFFSET RX_track_begin_entry     ; COMMAND = begin entry
        dw      0                               ; priority
        dw      OFFSET BRAVO_SEQUENCER          ; sequence pointer

RX_track_begin_entry:
        dw      DEF_request_entry               ; COMMAND = begin entry
        dw      DEF_pid_bravo                   ; dest. Proc ID (bravo)
        dw      offset Targsup_TCB              ; dest. Task ID (Targ_Sup)
        dw      offset Targsup_TCB.ENTRY1_Q     ; entry = Get_Target_Msg
   ;reply
        DEF_alpha_address <>
        DEF_bravo_address <>
        dw      OFFSET RX_track_end_entry       ; COMMAND = accept complete
        dw      0                               ; priority
        dw      OFFSET ALPHA_SEQUENCER          ; sequence pointer

RX_track_end_entry:
        dw      DEF_accept_complete             ; COMMAND = accept complete
        dw      DEF_pid_alpha                   ; dest. Proc ID (alpha)
        dw      offset Track_TCB                ; dest. Task ID (Track)
                                                ; caller entry not required

ALPHA_SEQUENCER dw      0
BRAVO_SEQUENCER dw      0


;
```

```
;  For now, all tasks have at most two entries, therefore this is
;  static.
;
;   The TCB contains a Synchronize semaphore which is used to
;   suspend itself and wait for a signal from another task.
;   This is followed by a list of entry records.  Each entry record
;   contains:
;    - A waiting flag used by the accepting task to indicate that it
;      has suspended waiting for a call on this entry (and possibly
;      others).
;    - A buffer List Pointer, This points to the buffer descriptor
;      for the first caller to this entry.  The buffer descriptor
;      provides the actual buffer address and a link to the next
;      descriptor.  This provides the FIFO queue for each entry.
;
;  In the case of the MAIN program, Entry1 is used for elaboration
;  control of library units.
;
TCB     struc
        SYNC_SEMAPHORE          DW      3 dup (0) ; to synchronize rendezvous
        ENTRY1_Q                DW      2 dup (0) ; waiting flag, next ptr
        ENTRY2_Q                DW      2 dup (0) ; waiting flag, next ptr
        REPLY_PTR               DW      0         ; pointer to replies
TCB     ends

Main_TCB        TCB     <>

Targsup_TCB     TCB     <>

Rocksup_TCB     TCB     <>

Guidebuf_TCB    TCB     <>

Reportbuf_TCB   TCB     <>

Track_TCB       TCB     <>

Control_TCB     TCB     <>

        end
        page    55,132
        TITLE   RTE - Distribted Ada Runtime Module-
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; FILE:  DA_RTE.ASM                                                    ;
;                                                                      ;
; RTE -  DISTRIBUTED Ada RUNTIME MODULE                                ;
;                                                                      ;
;;;;;;;;;;;;;;;;;;;;;; Distribution and Copyright ;;;;;;;;;;;;;;;;;;;;;;
;   Derivation    : LabTek Distributed Ada V1.0                        ;
;                                                                      ;
;   This Distributed Ada Runtime inherits the LabTek copyright.        ;
;   The following copyright must be included in all software           ;
;   utilizing this Ada Runtime.                                        ;
;                                                                      ;
;   Copyright 1989 by LabTek Corporation, Woodbridge, CT, USA          ;
;                                                                      ;
;   Permission to use, copy, modify, and distribute this               ;
;   software and its documentation for any purpose and without         ;
;   fee is hereby granted, provided that the above copyright           ;
;   notice appear in all copies and that both that copyright           ;
;   notice and this permission notice appear in supporting             ;
;   documentation, and that the name of LabTek not be used in          ;
;   advertising or publicity pertaining to distribution of the         ;
;   software without specific, written prior permission.               ;
;   LabTek makes no representations about the suitability of           ;
;   this software for any purpose.  It is provided "as is"             ;
;   without express or implied warranty.                               ;
;                                                                      ;
;;;;;;;;;;;;;;;;;;;; Disclaimer ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                                                      ;
;   This software and its documentation are provided "AS IS" and      ;
;   without any expressed or implied warranties whatsoever.            ;
;   No warranties as to performance, merchantability, or fitness       ;
;   for a particular purpose exist.                                    ;
;                                                                      ;
;   In no event shall any person or organization of people be          ;
;   held responsible for any direct, indirect, consequential           ;
;   or inconsequential damages or lost profits.                        ;
;                                                                      ;
;;;;;;;;;;;;;;;;;;;;; END-PROLOGUE ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                                                      ;
; Runtime Code to implement prototype Distributed Ada Services         ;
;                                                                      ;
; This module implements the remote rendezvous operations to           ;
; support distributed Ada.                                             ;
;                                                                      ;
; Currently provided are:                                              ;
;   Remote_Entry, Remote_Select, Remote_Accept, Remote_End_Accept,     ;
;   Remote_Elab_Start, Remote_Elab_Wait, Remote_Elab_Continue.         ;
;   Local_End_Accept                                                   ;
;                                                             ;
; Ver Date     Description                                    ;
;                                                             ;
; 0.1 Nov-88 : Initial prototype                              ;
;                                                             ;
;                                                             ;
;                                                             ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
        .model  large

        public  Initialize
        public  Remote_Entry, Local_Entry, Any_Select
        public  Remote_Accept, Remote_End_Accept, Local_End_Accept
        public  Remote_Elab_Start, Remote_Elab_Wait, Remote_Elab_Continue         public  NET_Receive             ; calle

; Vendor Runtime Services
        extrn   VRTIF_Wait:far          ; Vendor Supplied P Semaphore operation
        extrn   VRTIF_Signal_I:far      ; Vendor Supplied V operation/interrupt
        extrn   VRTIF_Signal:far        ; Vendor Supplied V operation
; Network IO Services
        extrn   TX_READY:near           ; Transmit ready semaphore
        extrn   IO_XMIT:near            ; Start transmission routine
        extrn   IO_Network_Init:near
        extrn   IO_ALLOCATE:near        ; allocate a buffer
        extrn   IO_DEALLOCATE:near      ; deallocate a buffer

        include DA_DEF.ASM              ; system definitions


cseg    segment common
        assume  cs:cseg,ds:cseg,es:cseg
        org     600H
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
;  Initialize    -- no parameters
;
Initialize:
        call    IO_Network_Init
        retf
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
;  ELABORATE_START:
;  This routine is called by MAIN to elaborate library packages on
;  remote processors.  After sending the start message to the designated
;  processor, this routine suspends the current task (main) waiting for
;  the CONTINUE reply.
;
;  Inputs:  TASK_ID (of main)
;           Transmit Control pointer (pointer to appropriate start msg)
;  Outputs: AX contains the Result code of the elaboration (Zero if OK).
;
```

```
Remote_Elab_Start:
        push    bp
        mov     bp,sp
;
;  upon entry to elaborate start, the parameters on the stack
;  provide the Transmit Control Ptr necessary to generate the start msg.
;
        call    IO_Xmit
        push    cs                      ; vendor Wait needs segment # too
        push    [bp+DEF_task_id]        ; first item in TCB is semaphore
        call    VRTIF_WAIT              ; wait for elaboration to complete
;
;  Now elaboration has completed, get result and deallocate receive
;  buffer
;
        mov     bx,[bp+DEF_task_id]     ; get my TCB
        add     bx,DEF_TCB_EID          ; point to entry Q 1
        call    REMOVE                  ; pull entry of queue (BX ptr)
        push    word ptr [bx+data_field]; fetch completion status
        call    IO_DEALLOCATE           ; pointed to by BX
        pop     ax                      ; get completion status into AX
        pop     bp
        retf

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
;   ELABORATE_WAIT : Suspends a task until it is given the signal
;                    to continue from the elaborating processor.
;
;   Go to Sleep on the Semaphore of the calling task.
;       INPUTS:
;               TASK_ID
;               ENTRY_ID
;
Remote_Elab_Wait:
        push    bp
        mov     bp,sp

;
;   First, check to see if there is already a message waiting on
;   the specified entry...
;
        mov     si,[bp+DEF_Entry] ;
        pushf                   ; preserve interrupt status
        cli                     ; go atomic
        test    word ptr [si+DEF_Next_Ptr],0FFFFH ; see if anything on entry
        jnz     REW010          ; go on if no wait necessary
        mov     word ptr [si],1 ; set WAITING Flag
        push    cs              ; vendor P routine needs segment
        push    word ptr [bp+DEF_task_id]
        call    VRTIF_WAIT      ; this also reenables interrupts
```

```
REW010:
;
;   After resumption, the wakeup message is on the main task's entry1,
;   it will be processed later during the elaborate continue procedure.
;
        popf                    ; restore interrupt status
        pop     bp
        retf


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
;   ELABORATE_CONTINUE : Notifies the elaborating processor that
;                        the specified elaboration in complete and
;                        that it can continue elaboration of other
;                        units.
;
;    Send "End_Elaborate" message to the processor that allowed
;       the elaboration to continue, as determined by the start msg
;       still queued to Mains entry1
;
;       Inputs: Task_ID
;               ENTRY_ID    on entry, changed to transmit control pointer
;               number of parameters : always ZERO
Remote_Elab_Continue:
        push    bp
        mov     bp,sp
        mov     bx,[bp+DEF_Entry] ; fetch entry id
        call    REMOVE          ; pull buffer descriptor off entry queue
        mov     si,[bx]         ; fetch actual buffer address into SI
        mov     si,[si+RCP]     ; get receive control pointer from rcv buffer
;
;  Now we are done with the received buffer, release it
;
        call    IO_Deallocate   ; release buffer, descriptor in BX
        lea     ax,[si+DEF_reply_offset]    ; fetch reply message pointer
        mov     [bp+DEF_TCP],ax ; put on stack for XMIT_IO in Transmit control
        call    IO_Xmit         ; transmit message pointed to by BX
        pop     bp
        retf

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
;   Remote_Entry
;
;   Send "Request_Entry" message with IN parameters
;   Wait on Entry_Wait_Semaphore
;   Copy OUT parameters
;   Release Buffer
;
```

```
;   PARAMETERS:         TASK_ID
;                       TRANSMIT CONTROL_PTR
;                       NUM IN PARAMS
;                       IN_PARAM1    (offset,segment,length)
;                        ...
;                       IN_PARAMn    (offset,segment,length)
;
;   NOTE: Stack Parameters are removed by caller
;
Remote_Entry:
        push    bp
        mov     bp,sp

        call    IO_Xmit
;
;   Now wait for Accept Complete to wake up
;
        push    cs
        mov     ax,[bp+OEF_task_id]     ; point to parameters
        push    ax                      ; offset of semaphore (TCB)
        call    VRTIF_Wait              ; go to sleep

        pop     bp
        retf

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
;   Local_Entry  : This routine is called for an entry of a task
;                       which is local (same processor) as the caller
;               Inputs: SRC-TASK_ID
;                       DST-TASK_ID
;                       DST-ENTRY_ID
;                       PARAMTER ADDRESS (OFFSET,SEGMENT)
;   Although the task is local to the caller, an IO buffer is allocated
;   to store the necessary pointers required by accepting tasks.  This
;   is later deallocated as part of the local_end_accept routine.  The
;   calling task is always suspended, and if the accepting task is "waiting"
;   it is signaled to wake up.
;
src_task_id     equ     6
dst_task_id     equ     8
dst_entry_id    equ     10
entry_param     equ     12

Local_Entry:
        push    bp
        mov     bp,sp
        call    IO_Allocate             ; get a buffer descriptor ptr in BX
        mov     si,[BX]                 ; fetch buffer address
;
;   currently only one parameter is used (either in or out).  Take advantage
;  of this to simplify interface to accepting task. The address of the
;  data area is provided in the first part of the buffer.  NOTE: this address
;  is backwards (segment=low address, offset=high address).
;
        mov     ax,[bp+entry_param]     ; get offset of param
        mov     [si+2],ax
        mov     ax,[bp+entry_param+2]   ; and segment
        mov     [si],ax
        mov     [si+4],bx               ; stuff buffer descriptor in buffer too
        mov     ax,[bp+src_task_id]     ; fetch our task id
        mov     [si+DEF_local_task_id],ax ; and put in calling task id there
        mov     si,[bp+dst_entry_id]    ; fetch entry queue head
;
; ATOMIC action follows...  Queue entry, if waiting signal acceptor
;
        pushf
        cli
        call    INSERT                  ; place buffer descriptor on entry Q
        mov     si,[bp+dst_entry_id]    ; fetch entry address again
        test    word ptr [si],0FFFFH    ; see if WAITING
        jz      le010                   ; go on if not
;
; server is waiting on accept, signal it
;
        mov     si,[bp+dst_task_id]     ; get task id

        mov     word ptr [si+DEF_TCB_EID],0  ; clear (all) waiting flags
        mov     word ptr [si+DEF_TCB_EID+DEF_TCB_ENTRY_SIZE],0 ; get other entry
        push    cs                           ; segment of semaphore
        mov     ax,[bp+dst_task_id]
        push    ax
        call    VRTIF_Signal            ; wake up server (may preempt ourselves)
;
; NOTE: This is the end of the atomic region (above Vendor runtime call
;       reenables interrupts!
;
le010:
        popf                            ; restore interrupt level
        push    cs                      ; now try to suspend ourselves
        mov     ax,[bp+src_task_id]
        push    ax
        call    VRTIF_Wait              ; may not suspend if server is higher
                                        ; priority and has already signaled us!
        pop     bp
        retf


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;  Any_Select
```

```
;
;   Check to see if any of the entries have callers. If not,
;   set the "Waiting" Flag in each of them, and go to sleep.
;   If one entry has a queued request, accept it and return
;   offset for "Case" table in DI and Entry pointer in CS:BX.
;
;   Note that this routine provides the synchronization and buffer
;   management functions only.  Separate code in the "LINK"
;   module is necessary to facilitate parameter transfer.
;
;   Parameter Sequence:
;      BP+6 =>  Task_ID           (Always Current Task)
;        +8     Number of open select alternatives
;               Entry_ID #1
;                  ...
;               Entry_ID #n
;
;   PARAMETERS ARE REMOVED FROM STACK BY CALLER
;
;       Output Parameters:   SI - selection entry id
;                            ES - data segment
;                            BX - data offset
entry_count     equ     8               ; offset from BP for entry count
entry_list      equ     10              ; offset from BP for entry list start

Any_Select:
        push    bp
        mov     bp,sp

;
;   ENTER CRITICAL REGION (cannot allow task to go on an entry queue
;   after we have checked it, but before setting waiting flag.
;
        pushf
        cli
;
;  First check each entry to see if any has a caller...
;
Rem_Sel00:                              ; will come back here after resume
        mov     cx,entry_count[bp]      ; number of entries
;
;  normaly might want to put OR CX,CX  ; JZ raise program_error  %%
;  since all entries are closed!  %%
;
        mov     si,0                    ; entry index
Rem_Sel10:
        mov     di,entry_list[bp+si]    ; get next entry ID
        test    word ptr [di+DEF_Next_Ptr],0FFFFH ; see if an caller is queued
        jnz     Rem_Sel50               ; if caller is present, go select
        add     si,2                    ; go to next entry in list
        loop    Rem_Sel10
```

```
;
;   all of the Entry Queues are Empty, mark each Waiting flag
;   and go to sleep.
;
        mov     si,0                    ; reset pointer
        mov     cx,entry_count[bp]      ; fetch number of entries again
Rem_Sel30:
        mov     di,entry_list[bp+si]    ; fetch entry id
        mov     word ptr [di],1         ; set WAITING flag
        add     si,2                    ; goto next entry
        loop    Rem_Sel30
;
;   The following runtime call will suspend this task, when it
;   resumes, the interrupt flag will be set again, and presumably,
;   one of the entries will have a caller queued.
;
        push    cs                      ; push segment of wait_semaphore
        push    [bp+DEF_task_id]        ; push offset  of wait_semaphore taskid
        call    VRTIF_Wait              ; do wait on semaphore
;
;   Now clear all the waiting flags
;
        cli
        mov     si,0                    ; reset pointer
        mov     cx,[bp+entry_count]     ; fetch number of entries again
Rem_Sel40:
        mov     di,entry_list[bp+si]    ; entry list on stack
        mov     word ptr [di],0         ; clear waiting flag
        loop    Rem_Sel40

        jmp     Rem_Sel00               ; go back and find caller

;
;   There is a caller on this entry queue, do start accept
;   Fetch the Caller's buffer, which has a (backward) pointer to
;   the parameter data
;
Rem_Sel50:
        mov     di,[di+DEF_Next_Ptr]    ; get buffer descriptor
        mov     bx,[di]                 ; fetch buffer address
        mov     ax,si                   ; get selector into AX
        shr     ax,1                    ; get count
        inc     ax                      ; to be compatible with VRTIF
        popf                            ; restore interrupt status
        pop     bp
        retf                            ; parameters are removed by caller

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
;   Remote Accept (TASK_ID, ENTRY_ID) return ES:BX_Param_Pointer
;
```

```
;    Simple Accept, see if someone on entry queue, if so
;    return with pointer to buffer in ES:BX, otherwise set
;    "Waiting" Flag and go to sleep on semaphore.
;
;    Inputs:  TASK_ID, ENTRY_ID
;
;    Outputs: Returns ES:BX pointing to Parameter Data List
;

Remote_Accept:
        push    bp
        mov     bp,sp

        mov     si,[bp+DEF_Entry]     ; fetch address of entry Q
        pushf                         ; save interrupt status
        cli                           ; go atomic
        test    word ptr [si+DEF_NEXT_PTR],OFFFFH ; if Zero, then list is empty
        jnz     RA010                 ; if caller is there, take it!
;
;   No caller on entry queue.  Set waiting flag and go to sleep
;
        mov     word ptr [si],1 ; set flag
        push    cs              ; push segment of my task semaphore
        mov     ax,[bp+DEF_task_id] ; get address of TCB
        push    ax
        call    VRTIF_WAIT      ; go to sleep waiting for caller
;  NOTE after vendor runtime call - interrupts are enabled!
;
;  Now Something is on the queue, provide address of data area in
;  ES:BX and return to caller.
;
RA010:
        popf                          ; restore interrupt status
        mov     si,[bp+DEF_Entry]; get entry address again
        mov     si,[si+DEF_NEXT_PTR] ; get address of buffer descriptor
        mov     bx,[si]         ; fetch actual buffer address
        mov     ax,cs           ; provide caller with segment of data
        mov     es,ax
        mov     [bx],ax         ; put data address as first words in buffer
        lea     ax,[bx+data_field] ; but written backwards
        mov     [bx+2],ax
        pop     bp
        retf

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
;    Remote_End_Accept
;    Send output parameters to caller.
;    Release buffer used to hold input (and output for now) parameters.
;
Remote_End_Accept:
        push    bp
        mov     bp,sp

        mov     bx,[bp+DEF_Entry]       ; fetch entry queue pointer
        call    REMOVE                  ; pull it off queue, leaving BX = BUFPTR
        push    bx                      ; save buffer descriptor pointer
        mov     si,[bx]                 ; fetch actual buffer address into SI
        mov     si,[si+RCP]             ; get RCP from rcv buffer
        lea     ax,[si+DEF_reply_offset]; fetch reply message pointer
        mov     [bp+DEF_TCP],ax         ; replace ENTRY on stack with TCP
        call    IO_Xmit                 ; transmit reply
;
;  Now we are done with the received buffer, release it
;
        pop     bx                      ; get descriptor ptr back
        call    IO_Deallocate           ; release buffer, descriptor in BX
        pop     bp
        retf


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
;    Local_End_Accept
;    Allow caller to continue (Note: this is for entry calls with parameters
;    that are all passed by reference. No copy-back is required).
;    All entries whether remote or local use a buffer, therefore deallocate
;    it when complete.
;
Local_End_Accept:
        push    bp
        mov     bp,sp

        mov     bx,[bp+DEF_local_entry_id] ; fetch entry ID
        call    REMOVE                  ; pull entry off queue BX now @ buffer
        mov     si,[bx]                 ; get buffer address
        push    cs                      ; push segment of semaphore
        push    word ptr [si+DEF_local_task_id] ; push calling Task's ID
        call    IO_Deallocate           ; deallocate buffer @ BX
        call    VRTIF_Signal            ; signal task to continue

        pop     bp
        retf


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;  Net_Receive - processes an incoming message          ;
;                                                       ;
;  This routine is called by the interrupt handler (in  ;
;  the IO Module) to initiate action based on the       ;
;  receipt of a packet.  When the service handler is    ;
;  called, BX contains the address of the buffer        ;
; descriptor, SI contains the RECEIVE CONTROL POINTER. ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

Net_Receive:
        mov     si,[bx]                 ; get address of actual buffer
        mov     si,[si+RCP]             ; Receive control pointer
        mov     di,[si+DEF_cmd_offset]  ; fetch command
        and     di,vector_mask          ; mask for valid vector values
        jmp     vector[di]
;
;   The following vector table implements the 'case' statement
;   on the message ACTION Field
;
vector_mask     equ     00000110B   ; valid range 0 - 6
vector  label   word
        dw      offset  Begin_Elaborate
        dw      offset  End_Elaborate
        dw      offset  Request_Entry
        dw      offset  Accept_Complete
;
;  Future versions of the vector table will include
;        Begin_Remove_Entry
;        End_Remove_Entry
;        Begin_Abort
;        End_Abort
;        Begin_Terminate
;        End_Terminate
;        Shared_Variable_Request
;       etc.
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;  This code section is executed upon receipt of a message initiating
;  a Begin_Elaborate request.  BX points to buffer descriptor.
;
Begin_Elaborate:
        mov     di,[bx]                 ; get data buffer
        mov     di,[di+RCP]             ; fetch receive control pointer
        mov     si,[di+DEF_eid_offset]  ; get entry ID
        call    INSERT                  ; put buffer on entry queue
        push    cs                      ; semaphore segment
        push    [di+DEF_tid_offset]     ; push task id of destination
        call    VRTIF_Signal_I          ; signal task to continue
        ret


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
;  This code section is executed upon receipt of a message initiating
; an End_Elaborate.  This message implies that the specified elaboration has
; been completed on the remote processor and elaboration can continue
; on the primary processor.
;
; INPUTS: BX points to buffer descriptor.
;
End_Elaborate:
;
        mov     si,[bx]                 ; get buffer address
        mov     si,[si+RCP]             ; fetch receive control pointer
        push    cs                      ; semaphore Segment
        mov     ax,[si+DEF_tid_offset]  ; semaphore Offset
        push    ax
        mov     si,[si+DEF_eid_offset]  ; get entry id offset
        mov     word ptr [si],0         ; clear WAITING FLAG
        call    Insert                  ; put on caller's queue
        call    VRTIF_Signal_I          ; signal task to continue
        ret

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
; This code section is executed upon receipt of a message initiating
; an entry call
;
; Place buffer on Entry queue, if "Waiting" for that entry is TRUE,
; then clear all Waiting Flags and signal Wait Semaphore.
;  INPUTS:
;     BX = Buffer descriptor pointer
;     SI = Receive control pointer (RCP)
;
Request_Entry:
        mov     dx,[si+DEF_tid_offset]  ; save task id for later
        mov     si,[si+DEF_eid_offset]  ; fetch entry id for later
        mov     di,[BX]                 ; fetch buffer address (again)
;
;  currently only one parameter is used (either in or out).  Take advantage
;  of this to simplify interface to accepting task.  The address of the
;  data area is provided in the first part of the buffer.  NOTE: this address
;  is backwards (segment=low address, offset=high address).
;
        lea     ax,[di+data_field]      ; get offset of parameter
        mov     [di+2],ax
        mov     ax,cs                   ; and segment
        mov     [di],ax
        mov     [di+4],bx               ; stuff buffer descriptor in buffer too
;
; ATOMIC action follows...  Queue entry, if waiting signal acceptor
;
        pushf
        cli
        mov     cx,[si]                 ; get WAITING flag for this entry
        call    INSERT                  ; place buffer descriptor on entry Q
        or      cx,cx                   ; test waiting flag
        jz      re010                   ; go on if not
;
;  server is waiting on accept, signal it
;
        mov     si,dx                   ; get task id
        mov     word ptr [si+DEF_TCB_EID],0  ; clear (all) waiting flags
        mov     word ptr [si+DEF_TCB_EID+DEF_TCB_ENTRY_SIZE],0 ; get other entry
        push    cs                      ; segment of semaphore
        push    si                      ; offset of semaphore (first in TCB)
        call    VRTIF_Signal_I          ; wake up server
;
re010:
        popf                            ; restore interrupt level
        ret                             ; return to interrupt handler

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;  ACCEPT_COMPLETE -
;  This code section is executed upon receipt of a message completing
;  an accept body (end rendezvous)
;
;  Post buffer containing Out Parameters and signal task to wake up
;
;   INPUTS:
;      BX = Buffer descriptor pointer
;      SI = Receive control pointer (RCP)
;
;
Accept_Complete:
        mov     si,[si+DEF_tid_offset]  ; fetch task id of caller
        mov     [si+DEF_TCB_REPLY],bx   ; provide caller with reply buffer
        push    cs                      ; push segment of caller semaphore
        push    si                      ; push offset of  same (TCB)
        call    VRTIF_Signal_I          ; wake up caller
        ret                             ; to finish interrupt

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                                                           ;
;  REMOVE - Remove Entry that is on entry queue                             ;
;                                                                           ;
;  Inputs:  BX points to entry Q                                            ;
;  Output:  BX points to buffer descriptor that was dequeued                ;
;                                                                           ;
;  All other registers are preserved                                        ;
;                                                                           ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
REMOVE:
        push    ax
        push    si
;
```

```
; do list operation as atomic action
;
        pushf
        cli
        mov     si,[BX+DEF_NEXT_PTR]     ; fetch buffer descriptor
        mov     ax,[si+DEF_NEXT_PTR]     ; get next buffer
        mov     [BX+DEF_NEXT_PTR],ax     ; update queue head
        popf
        mov     bx,si                    ; return pointer in BX
        pop     si
        pop     ax
        ret

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                                                            ;
;  INSERT - INSERT Entry onto the end of an entry queue                      ;
;                                                                            ;
;  Inputs:  SI ppints to entry Q                                             ;
;           BX points to buffer descriptor                                   ;
;                                                                            ;
;  Outputs: SI points to last entry on Q                                     ;
;                                                                            ;
;  All other registers are preserved                                         ;
;                                                                            ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
INSERT:
        push    ax
;
;  do list operation as atomic action
;
        pushf
        cli
INSERT10:
        mov     ax,[si+DEF_next_ptr]     ; get next buffer on entry queue
        or      ax,ax                    ; see if end of list
        jz     INSERT20                  ; end of list, go insert it
;
;  this is not end of list, keep searching
;
        mov     si,ax
        jmp     INSERT10
;
;  found spot on list, insert it
;
INSERT20:
        mov     [si+DEF_next_ptr],bx     ; put on end of list
        popf                             ; restore interrupt flag
        mov     bx,si                    ; return pointer in BX
        pop     ax
        ret
```

```
cseg        ends

        end
```

```
                    page    55,132
                    TITLE   Setup - Distributed Ada Network Initialization
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; FILE:  DA_SETUP.ASM                                               ;
; Distributed Ada - Setup                                           ;
;                                                                   ;
; This module initilizes the network to prepare for distributed     ;
; processing.                                                       ;
;                                                                   ;
;;;;;;;;;;;;;;;;;;;; Distribution and Copyright ;;;;;;;;;;;;;;;;;;;;
;  Derivation    : LabTek Distributed Ada V1.0                      ;
;                                                                   ;
;  This Distributed Ada Runtime inherits the LabTek copyright.      ;
;  The following copyright must be included in all software         ;
;  utilizing this Ada Runtime.                                      ;
;                                                                   ;
;  Copyright 1989 by LabTek Corporation, Woodbridge, CT, USA        ;
;                                                                   ;
;  Permission to use, copy, modify, and distribute this             ;
;  software and its documentation for any purpose and without       ;
;  fee is hereby granted, provided that the above copyright         ;
;  notice appear in all copies and that both that copyright         ;
;  notice and this permission notice appear in supporting           ;
;  documentation, and that the name of LabTek not be used in        ;
;  advertising or publicity pertaining to distribution of the       ;
;  software without specific, written prior permission.             ;
;  LabTek makes no representations about the suitability of         ;
;  this software for any purpose.  It is provided "as is"           ;
;  without express or implied warranty.                             ;
;                                                                   ;
;;;;;;;;;;;;;;;;;;; Disclaimer ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                                                   ;
;  This software and its documentation are provided "AS IS" and     ;
;  without any expressed or implied warranties whatsoever.          ;
;  No warranties as to performance, merchantability, or fitness     ;
;  for a particular purpose exist.                                  ;
;                                                                   ;
;  In no event shall any person or organization of people be        ;
;  held responsible for any direct, indirect, consequential         ;
;  or inconsequential damages or lost profits.                      ;
;                                                                   ;
;;;;;;;;;;;;;;;;;;; END-PROLOGUE ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                    .model  large
                    public  Setup

                    include DA_NW.ASM
cseg                segment     common
                    assume  cs:cseg,ds:cseg,es:cseg
                    org   0A00H
Setup:
```

```
                mov     dx,cntrl        ; Gate array controller
                mov     al,eth_enable_reset
                out     dx,al
                mov     al,eth_disable_reset
                out     dx,al
                mov     al,eth_access_prom
                out     dx,al
                mov     cx,6
                mov     ax,cs
                mov     es,ax                   ; set es:di to receive board
                mov     di,offset BOARD_ADDRESS ; address from prom
                mov     dx,prom_address_0
GET_ADDRESS:
                in      al,dx
                stosb
                inc     dx
                loop    GET_ADDRESS

                mov     dx,cntrl                ; select no-sharing adapter,
                mov     al,eth_recv_select      ; and external transceiver
                out     dx,al

                mov     dx,gacfr                ; 8K of memory mapped space,
                mov     al,eth_lan_config       ; with interrupts enabled
                out     dx,al

                mov     dx,dqtr                 ; # of bytes to transfer on
                mov     al,eth_rem_DMA_burst    ; a remote DMA burst (n/a)
                out     dx,al

                mov     dx,idcfr                ; interrupt IRQ and DMA
                mov     al,eth_irq_line         ; channel selection (DMA n/a)
                out     dx,al

                mov     dx,damsb                ; 8k configuration for remote
                mov     al,eth_rem_DMA_config   ; DMA. Not used, but minimum
                out     dx,al                   ; value needed

                mov     dx,pstr                 ; start of receive buffer.
                mov     al,eth_recv_buf_start   ; Value MUST match that in
                out     dx,al                   ; NIC_pstart

                mov     dx,pspr                 ; end of receive buffer.
                mov     al,eth_recv_buf_end     ; Value MUST match that in
                out     dx,al                   ; NIC_pstop

                mov     dx,NIC_cr               ; stop NIC activity
                mov     al,eth_nic_stop
                out     dx,al

                mov     dx,NIC_dcr              ; local DMA transfers as
```

```
            mov   al,eth_nic_DMA_config   ; 8 byte bursts
            out   dx,al

            mov   dx,NIC_rbcr0            ; remote DMA setup (remote
            mov   al,eth_remote_DMA_lo    ; DMA not used, only local
            out   dx,al                   ; used)

            mov   dx,NIC_rbcr1            ; hi byte of # of bytes to
            mov   al,eth_remote_DMA_hi    ; transfer during a remote
            out   dx,al                   ; DMA operation

            mov   dx,NIC_rcr              ; accept runt, errored, broad-
            mov   al,eth_packet_types     ; cast and multicast packets
            out   dx,al

            mov   dx,NIC_tcr              ; go into internal loopback
            mov   al,eth_nic_mode         ; mode to finish programming
            out   dx,al                   ; (see anomalies - p. 52)

            mov   dx,NIC_bndy             ; overwrite protection rgtr.
            mov   al,eth_bndy_start       ; (protects unread packets)
            out   dx,al

            mov   dx,NIC_pstart           ; start of receive queue
            mov   al,eth_recv_buf_start
            out   dx,al

            mov   dx,NIC_pstop            ; end of receive queue
            mov   al,eth_recv_buf_end
            out   dx,al

            mov   dx,NIC_isr              ; clear interrupt status
            mov   al,eth_int_status
            out   dx,al

            mov   dx,NIC_imr              ; enable interrupts
            mov   al,eth_ints_enabled     ; transmit interrupts are
            out   dx,al                   ; at bit 02h

            mov   dx,NIC_cr               ; access page 1 registers
            mov   al,eth_access_page_1
            out   dx,al

            mov   dx,phys_address_0        ; let NIC know its address
            mov   ax,cs
            mov   ds,ax
            mov   si,offset BOARD_ADDRESS  ; from the prom
            mov   cx,6                     ; number of addresses to give
GIVE_ADDRESS:
            lodsb
            out   dx,al
```

```
                    inc   dx
                    loop  GIVE_ADDRESS            ; load all addresses

                    mov   dx,NIC_curr             ; load current receive pointer
                    mov   al,eth_recv_buf_start   ; with pstart
                    out   dx,al

                    mov   dx,NIC_cr               ; access page 0 registers
                    mov   al,eth_access_page_0
                    out   dx,al

                    mov   dx,NIC_cr               ; start NIC chip
                    mov   al,eth_start_nic
                    out   dx,al

                    mov   dx,NIC_tcr              ; exit internal loopback mode
                    mov   al,eth_exit_mode
                    out   dx,al

                    mov   ax,net_memory_seg       ; initialize LAN memory to
                    mov   es,ax                   ; zeroes
                    mov   cx,net_memory_size/2    ; in words
                    xor   di,di                   ; start at begin of segment
                    cld
                    mov   ax,0000                 ; initialization value
FILL:
                    stosw
                    loop  FILL
                    ret

BOARD_ADDRESS       db    6 dup (?)               ; holds board address

cseg                ends

                    END
```

```
        page    55,132
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; FILE: DA_VRTIF                                                      ;
; Distributed Ada - Vendor Runtime Interface                          ;
; This module provides the addresses within the                       ;
; Vendor supplied runtime for required tasking primatives.            ;
;                                                                     ;
;;;;;;;;;;;;;;;;;;;;; Distribution and Copyright ;;;;;;;;;;;;;;;;;;;;;;
;  Derivation   : LabTek Distributed Ada V1.0                         ;
;                                                                     ;
;  This Distributed Ada Runtime inherits the LabTek copyright.        ;
;  The following copyright must be included in all software           ;
;  utilizing this Ada Runtime.                                        ;
;                                                                     ;
;  Copyright 1989 by LabTek Corporation, Woodbridge, CT, USA          ;
;                                                                     ;
;  Permission to use, copy, modify, and distribute this               ;
;  software and its documentation for any purpose and without         ;
;  fee is hereby granted, provided that the above copyright           ;
;  notice appear in all copies and that both that copyright           ;
;  notice and this permission notice appear in supporting             ;
;  documentation, and that the name of LabTek not be used in          ;
;  advertising or publicity pertaining to distribution of the         ;
;  software without specific, written prior permission.               ;
;  LabTek makes no representations about the suitability of           ;
;  this software for any purpose.  It is provided "as is"             ;
;  without express or implied warranty.                               ;
;                                                                     ;
;;;;;;;;;;;;;;;;;;; Disclaimer ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                                                     ;
;  This software and its documentation are provided "AS IS" and       ;
;  without any expressed or implied warranties whatsoever.            ;
;  No warranties as to performance, merchantability, or fitness       ;
;  for a particular purpose exist.                                    ;
;                                                                     ;
;  In no event shall any person or organization of people be          ;
;  held responsible for any direct, indirect, consequential           ;
;  or inconsequential damages or lost profits.                        ;
;                                                                     ;
;;;;;;;;;;;;;;;;;;; END-PROLOGUE ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

                public  VRTIF_I8259, VRTIF_vector_base
                public  VRTIF_Wait, VRTIF_Signal, VRTIF_Signal_I

VRTIF_I8259          equ     20H        ; address of interrupt controller chip
VRTIF_vector_base    equ     200H       ; base of vector table

vrtif           segment at 4000h
                org     3F56H
VRTIF_Wait      label   far      ; R1TESS7P P semaphore operation (non-interrupt)
```

```
                org     3F6CH
VRTIF_Signal    label   far     ; R1TESS?V V semaphore operation (non-interrupt)

                org     3F21H
VRTIF_Signal_I  label   far     ; R1TESI?VI V semaphore operation (interrupt)

vrtif           ends
                end
```

21 Appendix J - Key Control for the Border Defense System

Version 1.0

21.1 Purpose

This document describes the technique and procedures for loading and verifying encryption keys used by the Border Defense System (BDS).

21.2 References

Data Encryption Standard, U.S. Department of Commerce, National Bureau of Standards, FIPS publication 46, 1977 January 15.

21.3 Key Management

Encryption keys used with the BDS shall conform to the Data Encryption Standard (DES) format. These keys consist of a bit string of 56 binary digits. Keys shall be entered as 7 pairs of hexadecimal digits, each pair representing 8 bits of the string. A minimum of 1 space shall be entered between each pair. The pairs of digits shall be terminated by an ASCII carriage return. No other characters other than space, carriage return, digits 0 through 9 and letters A through Z (case insensitive) are permitted. Note that parity bits are not entered into the BDS (or related hardware) by users, but may be computed internally within the system to augment the 56 bit string. Keys shall be provided by the security officer prior to system initialization.

Key entry shall be via an ASCII keyboard interface in either upper or lower case characters. Keys shall be entered twice to verify that there are no errors during entry. Keyboard echo shall be disabled during key entry. BDS equipment shall verify keys are in the correct format and ensure that both key entries are identical.

Each BDS system component requiring a key, shall not become fully operational until after a conforming key has been entered. Key entry shall be accompanied by the following messages, as appropriate:

```
Prompt 1:
            ENTER ENCRYPTION KEY  :

Prompt 2: (only if Key entry 1 is valid)

            REENTER ENCRYPTION KEY:

Response to valid entries:

            KEY ACCEPTED

Response to invalid entries:

            KEY REJECTED, INVALID FORMAT
   or
            KEY REJECTED, VERIFICATION ERROR
```